VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Distribuované datové struktury pro masivně paralelní zpracování dat

Distributed Data Structures for Parallel Data Management

2013 Aleš Nedbálek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Aleš Nedbálek**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Distribuované datové struktury pro masivně paralelní zpracování dat
Distributed Data Structures for Parallel Data Management

Zásady pro vypracování:

S růstem objemu dat a rozvojem Internetu se zvyšuje potřeba distribuce data a masivně paralelního vyhledávání v těchto datech. Úkolem diplomové práce je zejména:

1. Seznámit se s distribuovanými datovými strukturami pro indexování vícerozměrných prostorů.
2. Vybranou datovou strukturu implementovat.
3. Provést experimenty a získané výsledky porovnat s výsledky stromových datových struktur.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **doc. Ing. Michal Krátký, Ph.D.**

Datum zadání: 18.11.2011

Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.*

V Ostravě 30. března 2013 .....

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. března 2013 .....

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. března 2013 .....

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli. Zvláště pak panu doc. Ing. Michalu Krátkému, Ph.D. za ochotnou pomoc a vedení při vypracovávání mé diplomové práce.

Abstrakt

Rostoucí trend zpracování velkého množství dat vede k distribuci zátěže na více výpočetních uzlů a vzniku škálovatelných distribuovaných datových struktur - SDDS. Rozložení dat umožňuje jejich paralelní zpracování, zvýšení propustnosti a duplicita dat mezi uzly může zajistit dostupnost při selhání. Těchto vlastností je třeba u aplikací s důrazem na dostupnost a s velkým počtem klientů.
V práci uvádíme shrnutí vlastností jednotlivých SDDS s popisem distribuce a rozložení dat mezi uzly. Uvedené struktury lze rozdělit podle použitého konceptu distribuce na lineárně hashované a stromové datové struktury.
V rámci vývoje navrhujeme podle pravidel dodržovaných SDDS vlastní koncept se způsobem distribuce a rozložení pohledu na data. Celý koncept je implementován v jazyce C++. Serializaci volání metod a komunikaci jsme zpočátku chtěli převzít z veřejně dostupných API knihoven. Následně jsme se však rozhodli pro vlastní implementaci. Navrhli a implementovali jsme metodu vzdáleného volání metod za pomoci dvou příkazů *Command* a *ResultSet*. Síťovou komunikaci testujeme na TCP a UDP protokolech. Datové struktury R-strom a B-strom k testům byly dodány z databázového systému QuickDB[24] implementovaného skupinou databázové systémy z katedry Informatiky. Implementace s sebou přinesla i mnoho problémů a různých řešení společně s testy (serializaci přístupu, síťové prostředí, vlákna).
Výsledkem implementace je aplikace vícevláknového serveru a klienta s možností využití pro různé datové struktury. Reálné použití našla aplikace v projektu SGS Detekce plagiovaných dokumentů. Přístup k aplikaci zajišťuje webový klient v ASP.NET. Testy síťové komunikace nám ukázaly omezení v propustnosti reálné sítě.
Na závěr jsme provedli testy vzniklé aplikace DDS a embedded řešení pro B-strom a R-strom. Bohužel se v testech projevil vliv virtualizace prostředí a nedostatek hardwarových prostředků. Nedosáhli jsme předpokládaných násobků propustnosti při replikaci dat. I přes tyto nesnáze jsou výsledky zajímavé. Při vkládání se projevilo snížení propustnosti s rostoucí replikací dat. Výsledky bodových dotazů poukázaly na úměrný růst propustnosti s počtem replikací a rozsahové dotazy se částečně přiblížily propustnosti embedded řešení.

Klíčová slova: lineární hash, stromové datové struktury, distribuované datové struktury, masivně paralelní zpracování dat, R-strom, B-strom

Abstract

The growing trend of processing large amounts of data leads to the distribution load among multiple nodes and creation of scalable distributed data structures - SDDS. The distribution of data allows parallel processing, increase throughput and duplication data between nodes can ensure availability in the case of failure. These properties are necessary for applications with an emphasis on accessibility and a large number of clients.
In this work we present the summary of each SDDS with a description of the distribution and data decomposition between nodes. These structures can be divided according to the concept used for linear hash and tree data structures.
Development suggested the rules and we followed them to create own concepts SDDS. Decomposition and the distribution of view on the data, we propose own solution. The whole concept is implement in the C++ language. Serializing a call method and communication we want take from publicly available API libraries. Then we decided for their own implementation. We have designed and implemented a method for remote method calls using two commands *Command* and *ResultSet*. Testing communication on TCP and UDP protocols. Data structures like R-tree and B-tree for testing were supplied. Implementation has also brought many problems and different solutions together with tests (serializing access, network environment, threads).
The result of the implementation is a multi-threaded server application and client enable to use various data structures. The real utilization found the application in to the project SGS Detection plagiarism documents. Access to the application provides a web client in ASP.NET. Tests of the network communication have shown us bandwidth constraints in a real network.
Finally, we conducted tests of SDDS and embedded solutions for the B-tree and R-tree. Unfortunately, demonstrated in tests virtualization environment and lack of hardware resources. We did not achieve the expected throughput with scalable data replication. Despite these difficulties the results are interesting. When inserting data we decreases permeability with increase data replication. Results of the point queries referred to the proportional grow throughput with the numbers of data duplicity and the range querys are quite approximate to throughput embedded solutions.

Keywords: Linear hash, Tree data structures, Distributed data structures, Massive parallel data management, R-tree, B-tree

Seznam použitých zkratek a symbolů

BATON	–	balanced tree overlay network
CPU	–	central processing unit
CRC	–	cyclic redundancy check
DH	–	distributed hashing
DDH	–	distributed dynamic hashing
DLH	–	dynamic linear hashing
DFS	–	distibuted file system
DAC	–	disk access cost
ECC	–	erasure correcting codes
GFS	–	Google file system
GF	–	Galois field
IAM	–	image adjustment message
IP	–	internet protocol
LH	–	linear hash
MTU	–	maximum transmission unit
MBR	–	minimum bounding rectangle
OID	–	object identification
RAM	–	random access memory
RPC	–	remote procedure call
RS	–	Reed-Solomon
SDDS	–	scalable distributed data structure
TCP	–	transmission control protocol
UDP	–	user datagram protocol
VPN	–	virtual private network
VLAN	–	virtual local area network

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Tento text pojednává o distribuovaných datových strukturách (DDS), způsobu rozdělení dat a udržování pohledu na toto rozdělení jednotlivými koncepty. Následně se v textu věnuji návrhu, implementaci a testování distribuovaného B-stromu a R-stromu.
V kapitole 2 pojednáváme o základních datových strukturách a jejich principu. Na základě znalostí z této kapitoly se dále věnujeme komplexnějším strukturám, tzv. škálovatelným distribuovatelným datovým strukturám SDDS [1].
V kapitole 3 uvádíme definici a základní principy pro docílení či přiblížení vlastností SDDS. Důvodů vzniku distribuovaných datových systémů a jejich využití je mnoho, např. paralelizace dotazů, rozložení zátěže a zajištění neustálé dostupnosti záznamů.
V práci popisujeme několik konceptů SDDS s jejich vlastnostmi a zmiňuje vzájemné rozdíly. Následuje odlišení distribuce a udržování konzistence dat mezi jednotlivými uzly DDS. Popisujeme i proces, jak klient dosáhne na určitý záznam umístěný v DDS. Distribuce pohledu na indexaci dat mezi uzly v síti a klienty je základem efektivní DDS. Jedná se o koncepty, které využívají rozdělení dat pomocí distribuované hash tabulky nebo stromové datové struktury.
Ze získaných poznatků vzniká návrh konceptu vlastní DDS v kapitole 4. Shrnujeme jednotlivé vlastnosti a dostupné technologie k navržení vhodné komunikace pro rozložení dat v síťovém prostředí. Přitom se pokoušíme dodržet co možná nejvíce kritérií definujících SDDS struktury. Koncept vznikal s ohledem na dostupné technické možnosti a implementaci pro testování.
Kapitola 5 popisuje starší a nově vzniklou komunikaci na způsob vzdáleného volání procedur (RPC). Pro přenos bylo nutné vytvořit serializaci jednotlivých parametrů volané metody a navrácené výsledky. Myslíme serializaci do řetězců na aplikační vrstvě komunikace TCP a UDP. Průběh implementace dal postupně vzniknout několika třídám serializujícím komunikaci mezi aplikací klienta a serveru. Komunikace je postavena na dvou typech zpráv `Command` a `ResultSet` s možností přenést servisní informace pro potřeby správy DDS.
Kapitola 6 se zabývá vývojem aplikace s úspěchy a neúspěchy první implementace. Z prvních chyb a požadavků na různorodé využití aplikace serveru vznikl diagram návrhu tříd. Byl použit způsob vyvažování pomocí zasílání pohledu klientovi na uzel, který je méně zatížen. Pokud nehovoříme jinak, pak uzlem nazýváme aplikaci serveru DDS umístěnou na fyzickém či virtuálním serveru. Navrhli jsme metodu pro výpočet a porovnání rozdílů v zátěži uzlů a inicializaci vyvážení zátěže. Po celou dobu vývoje se snažíme dodržet námi definovaná pravidla konceptu z kapitoly 4.
Bez vlastního zpracování komunikačních socketů na nejnižší vrstvě bychom nebyli schopni dosáhnout maximálního využití síťových prostředků. Vzniklou třídu `cSocket` zajišťující síťovou komunikaci popisujeme s jejími možnostmi, nastavením a výsledky testování na reálné síti v kapitole 6.3.
Kapitola 6.4 se zabývá popisem a řešením problémů, které nastaly během implementace jednotlivých tříd a následných testů. Jedná se o problémy spojené s alokací paměti, synchronizací, chováním aplikace klient/server a testovacím prostředím.

Průběh, výsledky a zhodnocení jednotlivých testů je uveden v kapitole 7. V testech se věnujeme srovnání dvou režimů. V prvním režimu embedded je databáze součástí klienta a je znemožněn víceuživatelský přístup za cenu vysoké propustnosti záznamů. Přesným opakem je režim klient/server dovolující přístup více uživatelů z několika počítačů najednou. Nejčastěji dnes užívaným řešením není embedded databázový systém, ale client/server bez distribuce dat. Počet operací je zde značně ovlivněn propustností sítě a nedosahuje tak výkonnosti embedded řešení. Navíc je zde nulová tolerance výpadku ze strany serveru. Vytvořením client/server s DDS se snažíme zbavit nevýhod obou řešení. Rozdělení hardwarových prostředků na jednotlivé uzly dnes hraje roli i v rámci cenové dostupnosti. Nechceme prověřit pouze funkčnost aplikace, ale nastínit i počet uzlů potřebný k dosažení vlastností aplikace srovnatelné s řešením embedded. Závěrečné zhodnocení a doporučení vyplývající z výsledků testů je uvedeno v kapitole 8.
Budoucnost a navržená vylepšení konceptu jsou uvedeny v závěru práce společně s vlastnostmi, které nebylo možné v rámci diplomové práce plně naimplementovat. Navrhujeme několik vylepšení aplikace pro dosažení vyšší škálovatelnosti, dynamičnosti a rychlosti. Zmiňujeme také vzniklé zdrojové kódy aplikace, které již našly využití i v jiných aplikacích a pracích.

2 Datové struktury

Tato kapitola popisuje tři datové struktury (DS), na nichž jsou postaveny distribuované datové struktury popsané v následující kapitole. Jako první popisujeme princip DS založených na lineárním hashování. Následují dvě struktury B^+-strom a R-strom zastupující stromové datové struktury.

2.1 LH*

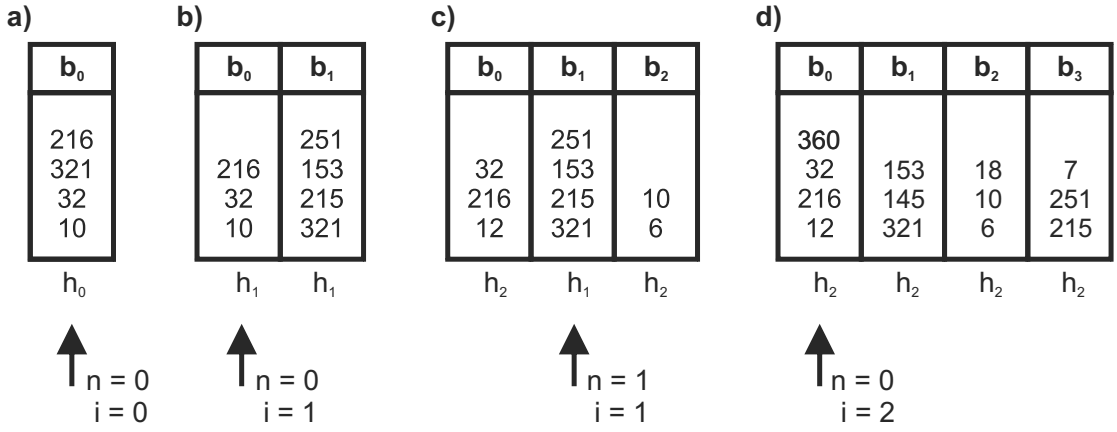
Základem všech LH* datových struktur je hashovací metoda pro rozšiřitelné disky nebo RAM soubory, které rostou nebo se zmenšují dynamicky, bez zhoršení využití prostoru nebo přístupového času. Soubory jsou uspořádány do bucketů (stránek) na disku nebo v RAM. Bucket je úložiště pro skupinu záznamů. První distribuovanou datovou strukturou SDDS byla LH* [1], po níž následovalo mnoho dalších zástupců.
LH* se skládá ze segmentů (záznamů) adresovatelných přes menší počet bucketů pomocí páru hash funkcí h_i a h_{i+1} na N*2^i adres. Kde N je počáteční počet záznamů v bucketu. Příkladem takovýchto funkcí je například modulo, viz níže.

$$h_i(\mathrm{C}) \rightarrow C \bmod N{*}2^i \tag{1}$$

Algoritmus pro adresování v LH*, tedy hash klíče C na adresu a, kde a = 0,1,2,... je následující [1]:

$a \leftarrow h_i$(C);
If $a < n$ then $a \leftarrow h_{i+1}$(C);

Struktura všech LH* konceptů se skládá ze základní jednotky - bucketu. Ten obsahuje jednotlivé záznamy, které se vkládají přes výpočet hash hodnot bucketu (i,n). Jednotlivé buckety jsou rozprostřeny mezi uzly DDS. Při vkládání expandují buckety ve chvíli, kdy bucket přeteče a je rozdělen na dva. Obsažené záznamy jsou rovnoměrně přerozděleny. Tento proces využívá funkci hash h_i (1), pokud je však překročena kapacita bucketu, je provedena funkce h_{i+1} (1). Zvláštní hodnota n, tzv. ukazatel slouží k určení, která funkce h_i nebo h_{i+1} bude aplikována na klíč objektu *OID*. Hodnota n roste s každou expanzí bucketů. Každé rozdělení přiřadí polovině objektů v bucketu n novou adresu n + N*2^i. Pokud je h_{i+1} přiřazeno všem bucketům, pak je i inkrementováno a celý proces začíná znovu.
Obrázek 1 zobrazuje lineární hashování pro buckety o maximální kapacitě 4 záznamy. Část **a)** znázorňuje první bucket b_0, na němž nastane kolize díky vložení záznamu 153. Kolize neboli přetečení bucketu je řešeno rozdělením dat mezi b_0 a b_1 s inkrementací hodnoty i pro hashovací funkci. Část **b)** ukazuje stav po rozdělení s vloženými záznamy 251 a 215. Hodnota n indikuje stále nejlevější bucket b_0 k rozdělení, k němuž dojde po vložení záznamů 6 a 12. Hodnota n je v části **c)** inkrementována a bucket b_0 s b_2 získají hashovací

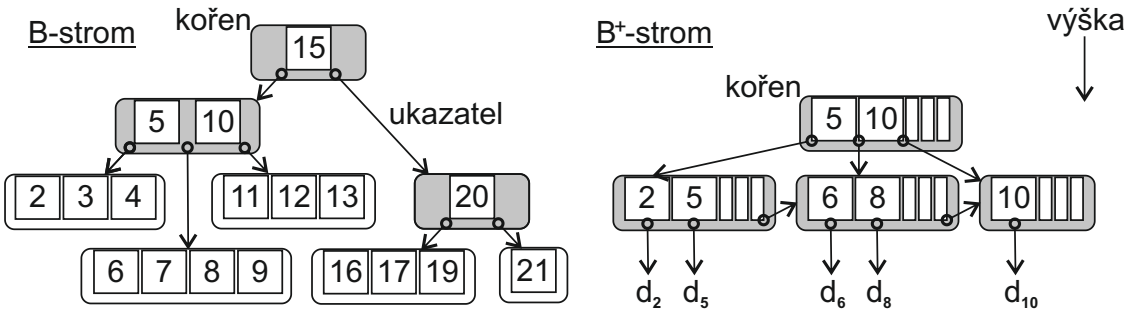


Obrázek 1: Ukázka lineárního hashování [1]

fci h_2. Ukazatel n se přesunul na b_1, jež má hashovací funkci nižší. Vložení záznamu 145 do b_1 vyvolá kolizi a způsobí vyrovnání hashovacích funkcí na h_2. Podle hodnoty $n = 1$, dojde k rozdělení b_1, přerozdělení záznamů a vzniku b_3. Dalším vkládáním záznamů 7, 360 a 18 zaplníme rovnoměrně buckety s vyhlídkou na budoucí přetečení b_0 ($n = 0$). Výsledný vzhled bucketů vidíme v části **d)**.
Výsledkem přerozdělení záznamů mezi buckety je konstantní přístup k záznamům a zatížení výkonu paměti, tato vlastnost je jedinečná pro LH*. Výkon přístupu k záznamu je tak jeden diskový přístup na vyhledání.
Při odebírání záznamů, mazání v LH*, dochází ke slučování bucketů, tato operace je inverzní k operaci rozdělení. Ukazatel n se dekrementuje a pokud je $n < 0$, bucket n splyne s posledním bucketem. Ke slučování by mělo docházet při malém vytížení bucketů. Regulace zátěže lze dosáhnout zavedením prahu T. Pokud není rozdělení řízeno, mělo by být T = 50 - 60%, avšak při kontrolovaném řízení může být T vyšší. Kontrolované řízení vyžaduje na každém uzlu funkci koordinátora řízení, nejedná se však o centrální prvek. LH* pracuje s jednou datovou strukturou, kterou rozloží mezi uzly. Zde se data rozkládají mezi jednotlivé datové struktury umístěné v uzlech. Na každém uzlu je tedy samostatná datová struktura, chcete-li databáze.

2.2 B$^+$-strom

B$^+$-strom [14] je stromová datová struktura vycházející z B-stromu umožňující rychlé vkládání, vyhledání a mazání dat. Využívá se v mnoha databázových systémech pro tvorbu tabulek s indexací primárních klíčů nebo pro ukládání dat (souborové systémy NTFS, JFS2, XFS a další). Oproti B-stromu jsou data uložena pouze v listových uzlech přístupných přes klíče od kořene stromu (viz obrázek 2).
Implementace stromu je založena na jednotlivých uzlech a jejich odkazech na potomky či data na disku. U B$^+$-stromu je díky konceptu propojení jednotlivých bloků a odkazů



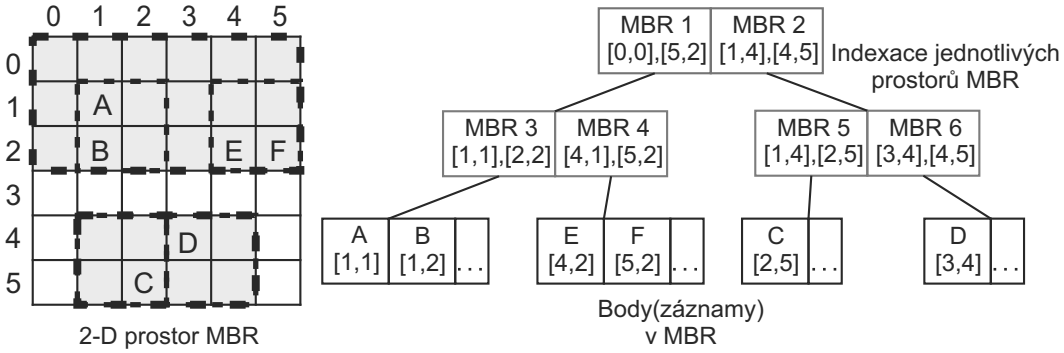
Obrázek 2: Ukázka B-stromu a B^+-stromu

na blok disku s daty dosaženo efektivního přístupu na disk. Výhoda propojení bloků je značná při rozsahových dotazech, tedy přístupu k souvislému bloku dat.
Uvádíme několik základních vlastností B^+-stromu, kde N je počet všech položek stromu, H je hloubka od kořene k listům a B je řád stromu:

- data jsou uložena pouze v listech
- všechny listy bez potomků jsou na stejné úrovni
- maximální možný počet záznamů je $N = B^H$
- místo pro uložení stromu je $O(N)$
- vyhledání potomka je v nejhorším případě $O(\log_B N)$ I/O přístupů

2.3 R-strom

Stromová datová struktura R-strom [14] pohlíží na data jako na body ve vícerozměrném prostoru. Jednotlivé body jsou shlukovány podle jejich vzájemné blízkosti/vzdálenosti, hovoříme tedy o vektorové datové struktuře. Body jsou indexovány v MBR.



Obrázek 3: Dvourozměrný prostor indexovaný pomocí R-stromu

Na obrázku 3 můžeme vidět příklad dvourozměrného prostoru a jeho indexaci na jednotlivé MBR. Hiearchie na obrázku obsahuje dva prostory MBR 1 a MBR 2, jež obsahují další MBR (3-6). Každý MBR je definován jako dvojice bodů ve vícerozměrném prostoru. Samotné indexované body se nacházejí na konci této hierarchie. Tato datová struktura podporuje jak rozsahový, tak bodový dotaz. Rozsahový dotaz je definován dvojicí bodů v prostoru a vrací jako výsledek všechny body nacházející se v tomto prostoru nazývaném dotazovací obdelník. Algoritmy vkládání a rušení bodů jsou složité a je jich více. V zásadě se při vkládání hledá prostor, kterému bod náleží. Pokud není nalezen, hledám vhodný prostor pro zvětšení a přidání tohoto bodu. K rozštěpení MBR může dojít při překročení kapacity a inverzně při rušení bodů ke sloučení dvou prostorů.

Definice 2.1 ***MBR*** *- minimální ohraničující obdelník, vyjadřuje oblast výskytu n-rozměrných objektů v rámci souřadnicového systému. Je vyjádřen dvojicí bodů Q_{low} a Q_{high} v prostoru.*

3 Škálovatelné distribuované datové struktury

Přirozeným vývojem v prostředí s rostoucím množstvím dat dalo vzniknout škálovatelným distribuovaným datovým strukturám (SDDS). V SDDS jsou data umístěna na různých uzlech. Jako první popisujeme základ konceptu struktur založených na lineárním hashování LH.
Kandidáty na využití SDDS s vysokou škálovatelností a propustností dotazů jsou dnes internetové aplikace, například vyhledávače a úložné prostory. Příkladem mezi vyhledávači je použití distribuovaného úložného systémů BigTable [15] pro správu rozsáhlých strukturovaných záznamů od společnosti Google. BigTable si udržuje tabulky v Google File System (GFS). Tento systém dokázal v roce 2005 dosáhnout 8 miliard indexovaných webových stránek. Strategii a techniku systému však Google úzkostlivě tají [18]. Stejně jako další firmy (Facebook, Amazon, YouTobe) s různými koncepty a techlologií SDDS.
Dnešní trend Grid computingu, cloudu a P2P aplikací vyžaduje vývoj směrem k distribuci a paralelismu. Hlavním faktorem bez ohledu na schopnosti jednoho procesoru nebo sítě mohou společné prostory (pool of sites) poskytnout více zdrojů a prostředků za mnohem nižší cenové náklady. Druhým faktorem je koexistence vysoké konektivity mezi klienty a servery. Vznikly pokusy o vytvoření škálovatelných distribuovaných datových struktur (SDDS) [1][6][12].

Definice 3.1 *SDDS je struktura definovaná nad dynamickou množinou uzlů, kde prostředí (síť serveru) splňuje:*

- *uzly sítě tvoří klienti se servery, ucelené stroje, nebo procesory s lokálními RAM v multiprocesorových strojích*
- *každý server poskytuje úložný prostor* F *pro objekty, charakter objektu je zde nedůležitý*
- *prostory rostou nebo se zmenšují dynamicky, bez zhoršení přístupového času*
- *velikost a počet úložných prostorů je neomezeně škálovatelný, možná redundance objektů zajišťuje obnovení a neustálou dostupnost všech objektů*
- *klienti vkládají a pracují s objekty pomocí OID (primární klíč), neví o ostatních klientech a klientem může být server*

Obecně hledáme strukturu splňující několik základních omezení, viz definice 3.2. Jedním z nich je postrádání centralizovaného adresáře uložených dat, a tedy i přímého výpočtu adres záznamů. V tomto důsledku je samostatná DS potencionálně účinnější. Nevyžaduje atomické aktualizace více klientů při vyhledávání, vkládání, rozdělení atd. [1].

Definice 3.2 *Strukturu splňující následující omezení lze nazvat Škálovatelnou distribuovanou datovou strukturou (SDDS) [8]:*

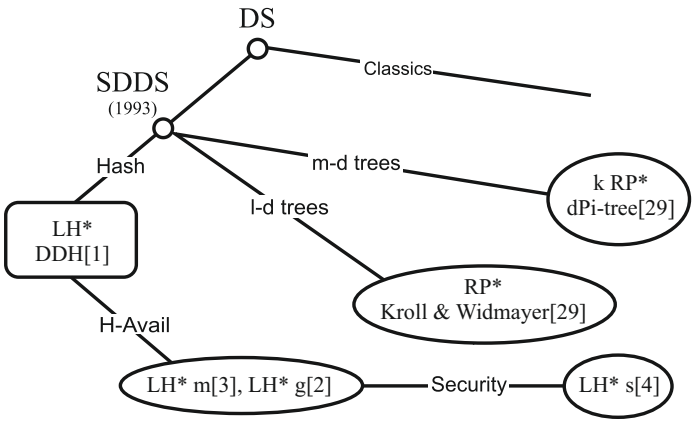
- *Data expandují na nové uzly pouze tehdy, když je již uzel efektivně zaplněn a využit.*

- *Není zde žádný hlavní uzel s výpočty objektových adres, např. pro přístup k centralizovaným DS, adresářům, bucketům.*
- *Přístup k souborům a údržba primitiv, např. vyhledávání, vkládání, rozdělení atd. nikdy nevyžaduje atomické aktualizace více klientů.*

Pokud chceme efektivní SDDS, musíme minimalizovat počty zpráv vyměňovaných prostřednictvím sítě, při současné maximalizaci výkonu. Tohoto cíle nelze dosáhnout v DDS založených na centralizaci.

3.1 Teorie DDS

Myšlenka rozdělení dat vedla na počátku vývoje DDS ke statickému režimu dělení. Jakmile se rozdělení ustálí, kritéria distribuce a počet míst zůstanou v životním cyklu systému statické. Ovšem aktualizacemi dat se tyto dva parametry mění. Změna a nárůst dat pak vyžaduje přerozdělení a vymazání staré kopie dat. Příkladem těchto systémů je $Round - robin$ [27], kde záznamy rovnoměrně rotují přes uzly. Pokud se záznamy přiřazují k uzlům pomocí hashovací funkce, jedná se o systém $Hash - declustering$ [26]. V $Range\ parttitioning$ [28] systému se naopak využívá rozdělení rozsahu hodnot klíčů mezi jednotlivé uzly. Výše jmenované systémy jsou staticky omezeny systémem rozdělení a adresace záznamů mezi uzly. Omezení statického režimu dalo vzniknout dynamickému režimu, který překonává například omezení rozšíření na více uzlů, než je zpočátku plánováno.



Obrázek 4: Známé rozdělení DS systémů [1]

Prvním takovýmto systémem byl DLH (dynamic linear hashing) [30], kde docházelo průběžně k atomickým aktualizacím parametrů všech uzlů. DLH využíval pevně spojené multiprocesory se sdílenou pamětí, data ukládal do paměti RAM a DS do lokální paměti. Toto schéma s pevně spojeným prostředím může být značně výkonné, avšak vylučuje distribuci dat pro škálování na větší množství uzlů. S cílem obejít toto omezení byly navrženy DDS založené na lineárním hashování LH*. Jednou z nich je DDH (distributed

dynamic hashing) [10] [11]- distribuované dynamické hashování. DDH využívá v základě dynamické hashování (DH), to je založeno na stromové hashovací funkci. Hlavní výhodou DDH je možnost okamžitého rozdělení přetékajících bucketů. Na rozdíl od DH si DDH neudržuje skutečný obraz celého stromu, pouze klienti si vytváří pomocí hashe obraz stromu, neboli *pohled* (view).
V kapitole 3.1.2 popisujeme distribuci pohledu na umístění dat mezi uzly v síti a způsoby jak jí dosáhnout. Jedná se o postupy, které využívají rozdělení dat pomocí distribuované hash tabulky nebo stromové datové struktury. Zvláštním případem je koncept LH, který distribuuje pouze hodnoty parametrů pro výpočet umístění.

3.1.1 Zrcadlení a proužkování

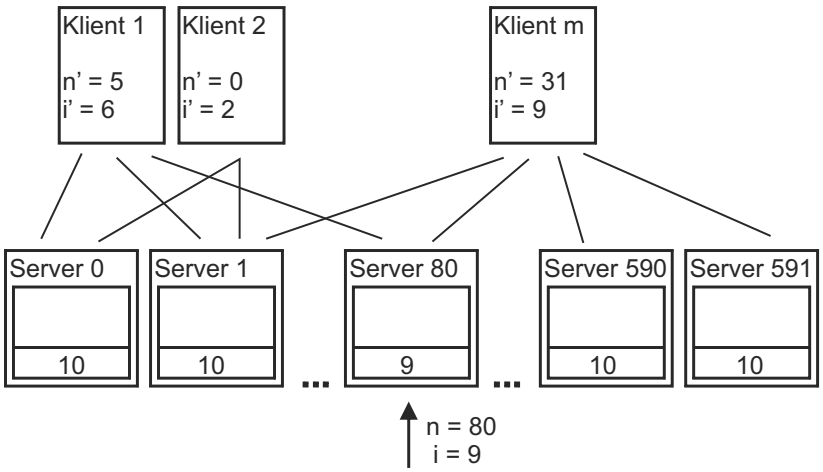
Mirroring, neboli zrcadlení, je technika pro zvýšení dostupnosti záznamů. Cenou za dostupnost je samozřejmě dvojnásobný požadavek na úložný prostor, viz obrázek 9. Každý záznam je dostupný i přes selhání jednoho uzlu a mnohdy i při vícečetném selhání. Tento koncept se využívá v aplikacích, které si nemohou dovolit nedostupnost záznamů, například v bankovnictví či u mobilních operátorů. SDDS využívající zrcadlení je uvedena v kapitole 3.2.2.
Segmentace záznamu (proužkování) je způsob, jak docílit zrcadlení záznamů společně s větší bezpečností. Záznam je rozdělen na $N > 1$ segmentů. Jednotlivé segmenty jsou rozloženy mezi uzly. Segmenty obsahují i paritní bity k ostatním segmentům. V případě nedostupnosti jednoho uzlu složíme celý záznam z paritních bytů ostatních dostupných segmentů. Pokud by někdo zachytil komunikaci uzlu s klientem, získá pouze segment celého záznamu (1/N). Oproti zrcadlení jsou nároky na uložiště menší. Blíže je průběh segmentace popsán v kapitole 3.2.3.

3.1.2 Pohled a jeho distribuce v DDS

V LH* se pomocí hashovací funkce vypočítá umístění bucketu se záznamem a IP uzlu, kde je bucket umístěn. Abychom záznamy rozdělili a posléze našli na jednotlivých uzlech, udržuje si klient i uzel pohled nad rozložením záznamů. Tento pohled může být tvořen polem, stromem či jinou strukturou s informacemi o rozmístění záznamů. Například, SDR-strom si drží pohled nad rozdělenými záznamy pomocí B-stromu. Klienti LH* využívají výpočet přes distribuované hodnoty hashovací funkce i a n. Pohled na rozložení záznamů nemusí být u klienta či uzlu úplný. Klienti se tak mohou dopouštět adresních chyb.

3.1.3 Distribuce pohledu v LH

S objekty na uzlech manipulují (hledají, vkládají) klienti pomocí klíče. Předpokládáme, že pro výpočet využijí správnou hodnotu i a n. Muselo by docházet k atomickým aktualizacím klientů, nebo k zřízení centrálním prvkem. Omezení SDDS režimu ani jednu z možností nedovoluje. Nevyžadujeme, aby pohled na i a n byl u klienta konzistentní. Klient postupuje v prvním kroku výpočtem adresy pomocí svých parametrů i' a n'. Ty

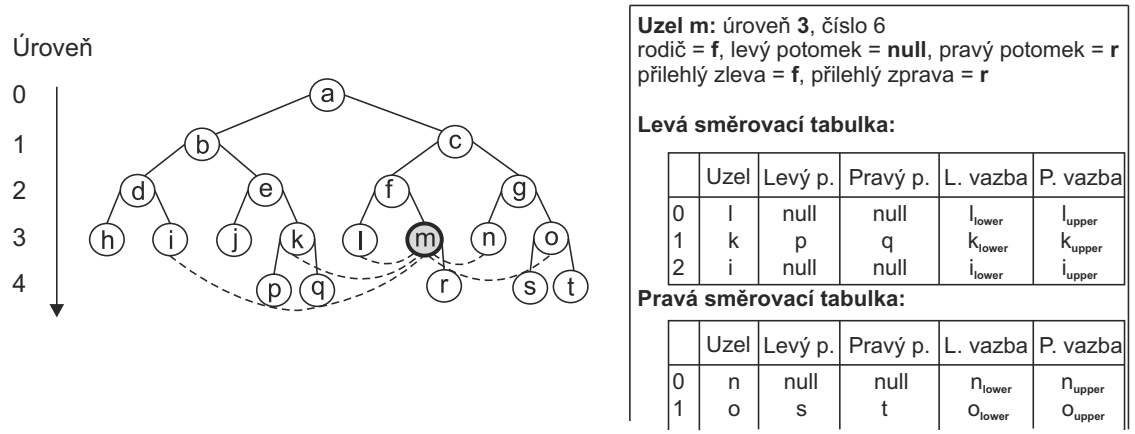


Obrázek 5: Koncept serverů (uzlů) a klientů v LH* [1]

jsou aktualizovány pouze po manipulaci klienta se záznamy v předchozí komunikaci.
Obrázek 5 ukazuje, že ani jeden klient nemá správnou představu o počtu bucketů SDDS. Pro výpočet adresy používáme hashovací funkci 1. Výpočet počtu udržovaných bucketů je tedy $2^i + n$. Například klient 1 vidí se svými hodnotami pouze 69 (2^6 + 5) bucketů, i když skutečný počet je 592 (2^9 + 80). Klienti se nevědomky dopustí při výpočtu *adresní chyby*. Druhý krok proběhne na straně uzlu, který přepočítá adresu vlastním výpočtem. Pokud hodnota bucketu nesedí, uzel přepočítá adresu a předá na ni klíč. Pokud druhý příjemce zjistí, že ani on není správným adresátem, přepočítá znovu adresu a přepošle klíč již na správný uzel. V nejhorším případě nastanou pouze dvě přeposlání. Ke komplikaci dochází, pokud se soubor zmenšil a klient odkazuje na již neexistující bucket. Jedním z řešení je odeslání aktuálních hodnot *i* a *n* uzlem klientovi pro nový výpočet adresy.
V neposlední řadě informujeme klienta, jenž vyvolal adresní chybu, o novém nastavení hodnot neboli obrazu IAM zprávou. IAM obsahuje první adresovanou úroveň souboru, tím se obraz klienta přiblíží skutečnému a předejdeme další adresní chybě klienta. Náklady na aktualizace přidané do odpovědí klientům jsou zanedbatelné.
Podle základní hashovací funkce a námi známého obrazu vypočte klient adresu *a'*. Samozřejmě se můžeme dopustit adresní chyby při výpočtu, to však příliš nevadí, neboť uzel vždy ověřuje obdržený klíč. V důsledku adresní chyby získá pomocí IAM zprávy aktualizaci obrazu stavu souboru a maximalizuje počet správně adresovaných klíčů. K aktualizaci využije hodnoty z IAM zprávy, adresu (kam byl klíč *C* poslán) a úroveň bucketu *j* na uzel *a*.
Každý uzel kontroluje přijatý klíč od klienta přes hodnotu své vlastní úrovně *j*. Soubory LH* neznají hodnotu n, a proto nemohou použít základní algoritmus výpočtu adresy. Místo toho přepočítají adresu klíče *C'*. Dokud nenastane sloučení bucketů a zmenšení adresního prostoru, nemůže nastat situace, při níž by klient vypočetl adresu mimo adresní prostor uzlů.

3.1.4 Pohled v BATON

Protokol BATON využívá distribuovanou stromovou datovou strukturu pro indexaci uzlů v síti. Narozdíl od konceptů s distribuovanou hash tabulkou podporuje rozsahové dotazy. Koncept je založen na indexaci uzlů pomocí binárního stromu. V síti s N uzly lze garantovat pro bodový i rozsahový dotaz nalezení odpovídajícího uzlu v O($logN$) krocích [21].
V každé úrovni stromu je uzel pojmenován svou pozicí ve stromě. Na obrázku 7 je uzel d na pozici $[2:0]$, m na pozici $[3:6]$. Pro uzel na pozici d je směrovací tabulka vyplněna uzly s pozicí $d-2^x$ pro $x \geq 0$, levá směrovací tabulka je vyplněna uzly s pozicí $d+2^y$ pro $y \geq 0$. Každý uzel si udržuje rozsah klíčů. Ve chvíli, kdy se připojí nový uzel, získá jako potomek od svého rodiče půlku rozsahu klíče, a tedy i záznamů. Lze tedy hledat ucelený prostor klíčů ve vzestupném pořadí, to umožňuje rozsahové dotazy.



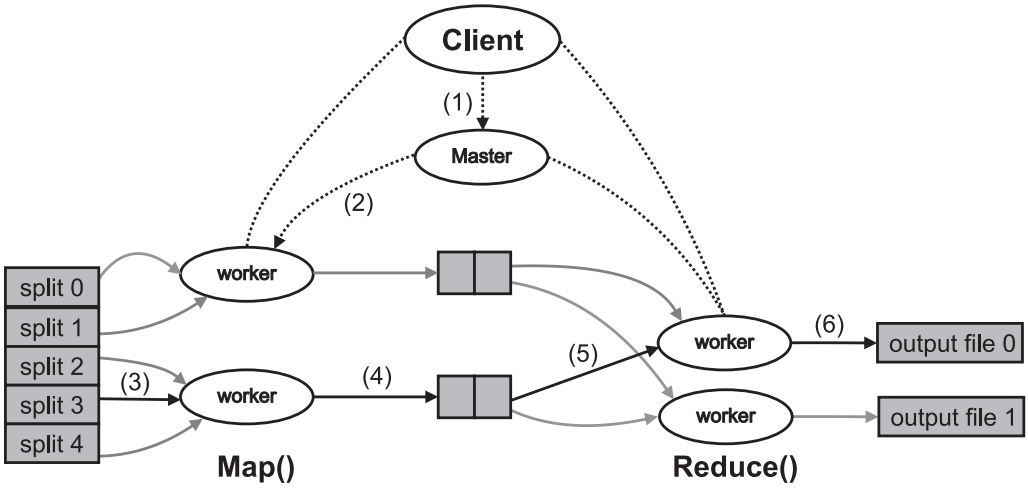
Obrázek 6: Architektura indexu binárního stromu v BATON [20]

Baton obsahuje dvě strategie pro vyvážení stromu. První z nich je přesunutí části zátěže na sousední uzel (vazební), pokud je oproti danému uzlu málo zatížen, přesunutím některých dat. Druhá strategie navazuje na první, pokud uzel nemá málo zatížený sousední uzel, hledá ho na síti. Uzel vyvolá proces, který se snaží v síti najít málo zatížený uzel. Následně nalezený uzel opustí své původní místo a přemístí se pod zatížený uzel jako potomek. Převezme část dat od rodiče a s ní i zátěže [21].
Bodový dotaz na klíč je v BATON vykonán pomocí směrování dotazu na nejvzdálenější uzly, dokud nenarazí na klíč. Pokud žádné takové směrovací uzly neexistují, použije se odkaz na rodiče, potomka nebo sousední uzel. Pro rozsahový dotaz Q najdeme první levě vázaný uzel Q_{low}. Proces dotazu pak prochází strom ve vzestupném pořadí až do dosažení horní meze Q_{up}.

3.1.5 (Ne)pohled v Map Reduce

Jedná se o koncept zpracování paralelních dotazů nad uzly. Klient se dotazuje na jeden z uzlů o němž ví. Nezohledňuje, zda uzel daný záznam vlastní či ne a dopouští se vědomě

adresní chyby. V tomto modelu si klient ani uzel netvoří pohled na rozložení záznamů v DDS. Případný dotaz na uzel vyvolá proces **Map()**, uzel dostane označení $Master$



Obrázek 7: Ukázka dotazu v konceptu Map Reduce (1-6)[17]

a přepošle dotaz všem ostatním známým uzlům - $worker$. Navrácené záznamy jsou následně zpracovány procesem **Reduce()**. Zredukují se veškeré redundantní záznamy a výsledek je zaslán zpět klientovi. Z hlediska počtu zaslaných zpráv a zátěže uzlů redundancí, zvláště při rozsahových dotazech, je tento koncept velmi náročný. Jedinou výhodou je přerozdělení zátěže na zpracování dotazu mezi uzly - $worker$ a odpadnutí veškerých nákladů spojených s pohledem na rozložení dat mezi uzly.

3.1.6 Další koncepty pohledu

Pro distribuci dat mezi uzly lze použít několik návrhů. CHORD, CAN, Pastry, and Tapestry jsou čtyři nejznámější P2P systémy. Každý implementuje distribuovanou hashovací tabulku, která je velmi efektivní při bodovém dotazu. Bohužel již nejsou vhodné pro rozsahové dotazy, nebot'uspořádání do hash tabulky rozbíjí posloupnost mezi daty. Tuto nevýhodu řeší návrh překrývající se stromové datové struktury BATON [20].

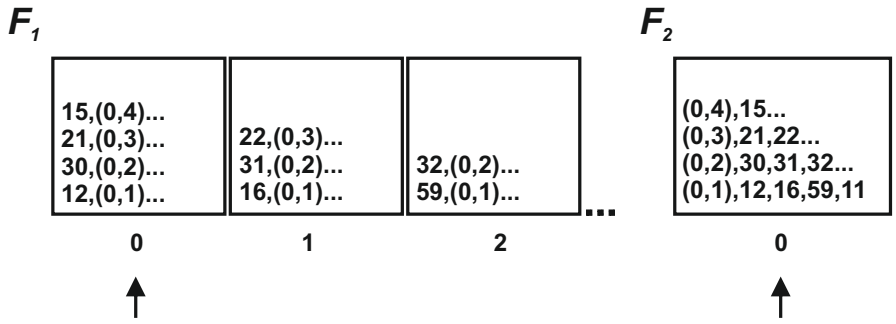
3.2 DDS

Existuje mnoho konceptů DDS s různými výhodami a tomu odpovídajícímu využití. V následujících podkapitolách popisujeme dvě skupiny zástupců DDS. První skupina je založena na lineárním hashování a druhá na stromových datových strukturách.
*LH*g* využívá paritní záznamy pro možnosti obnovení nedostupných záznamů [2]. *LH*m* zrcadlí záznamy pro větší dostupnost při selhání jednoho či více uzlů [3]. *LH*s* je navrženo pro větší bezpečnost s využitím *stripingu*, proužkování záznamů neboli segmentace [4]. Posledním ze zmíněné skupiny je *LH*rs*, jež podporuje nedostupnost jednoho ze svých uzlů s využitím takřka minimálního úložného paritního prostoru za pomoci

Reed-Salomon kalkulu [5].
Následují dva zástupci stromových DS, z nichž prvním je *CG-index* [15]. Jedná se o indexaci dat v prostředí cloudu za pomoci B^+-stromu, využívá se v mnoha databázových systémech a pro ukládání dat na disk v souborových systémech. *SDR-strom* [20] využívá principy organizace R-stromu a splňuje základní principy SDDS. SDR-strom vytváří binární strom mapovaný z několika nebo všech uzlů na síti.

3.2.1 LH*g

LH*g [2] je navržen pro vysokou dostupnost prostřednictvím nového principu uskupení záznamů. Dokáže rychle nahradit nedostupný záznam pomocí metody *hot spare*, neboli rychlé náhrady. Každý jednotlivý záznam může být obnoven za pomocí skupinového záznamu s paritou záznamu. Nepodporuje však možnost selhání více bucketů se záznamy najednou.
LH*g tvoří dvojice uzlů F_1 a F_2, viz obrázek 8. Koordinátor řídí primární uzel F_1 a paritní uzel F_2. Každý bucket m v F_1 má počítadlo vkládání záznamů r. Nově vložený záznam získá hodnotu skupiny a pořadí jeho vložení do bucketu ve tvaru (g, r). Unikátní označení skupiny je dáno výpočtem $g = Int(m/k)$. Vzniklé označení se nemění a zůstává od doby vložení do smazání stejné.
Pro každý klíč (g, r) existuje paritní záznam v F_2 s dvojicí neklíčových údajů. První hodnota je pole klíčů ve skupině $c_1 \ldots c_l$. Druhá hodnota obsahuje paritní bity z neklíčových částí záznamů v g. Paritní bit p_i je *XOR* všech i-tých bitů ze skupiny g. Obsah jednoho záznamu z g, lze obnovit za předpokladu, že jsou dostupné všechny ostatní záznamy ze skupiny g [2]. Pokud si představíme, jakým způsobem se postupně záznamy v bucketech



Obrázek 8: Ukázka struktury LH*g [2]

přesouvají dělením, zjistíme, že se žádné dva záznamy ze stejné skupiny g neocitnou ve stejném bucketu. Pro $k = 4$ by záznamy z bucketu 0 přešly postupně do bucketů 4,8,12. . ., ale záznamy z bucketu 1 by šly do bucketu 5,9,13. . .. *Tato vlastnost umožňuje obnovit záznam a popřípadě i všechny záznamy z nedostupného bucketu a uzlu.*
Obě zotavení má na starosti koordinátor, jenž inicializuje vyhledání všech členů skupiny

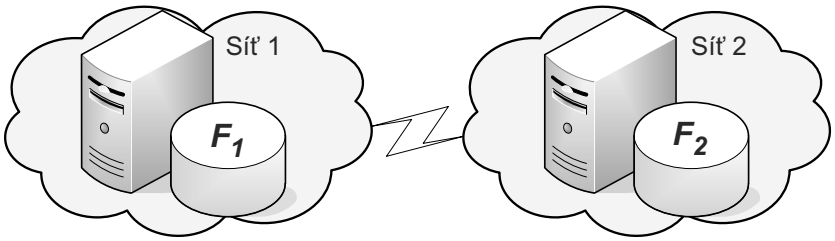
bucketu a vypočítá zpětně neklíčové části záznamu.
Nevýhodou tohoto režimu je zdvojnásobení zpráv při aktualizaci záznamu. Zasíláme zprávu i paritnímu bucketu pro přepočet paritních bitů. Dále vzniknou i velké náklady na zprávy při obnovení, kdy se vyhledávají a posílají všechny záznamy ze skupiny g.

3.2.2 LH*m

LH*m [3] splňuje definici zrcadlení 3.3. Výhodou tohoto řešení je transparentnost na úrovni LH*, kde pro každý klíč existuje jedna adresa bucketu. Nevýhodou jsou náklady na hardwarové vybavení serveru a snížená ochrana proti katastrofické chybě selhání sítě s právě zrcadlenými servery.

Definice 3.3 *Koncepty založené na zrcadlení by měly splňovat některé následující vlastnosti:*

- *dostupnost všech záznamů i přes selhání jednoho uzlu*
- *dostupnost všech záznamů ve většině případů selhání více uzlů*
- *efektivní zotavení po selhání*
- *vyvážení zátěže (load-balancing) po selhání*

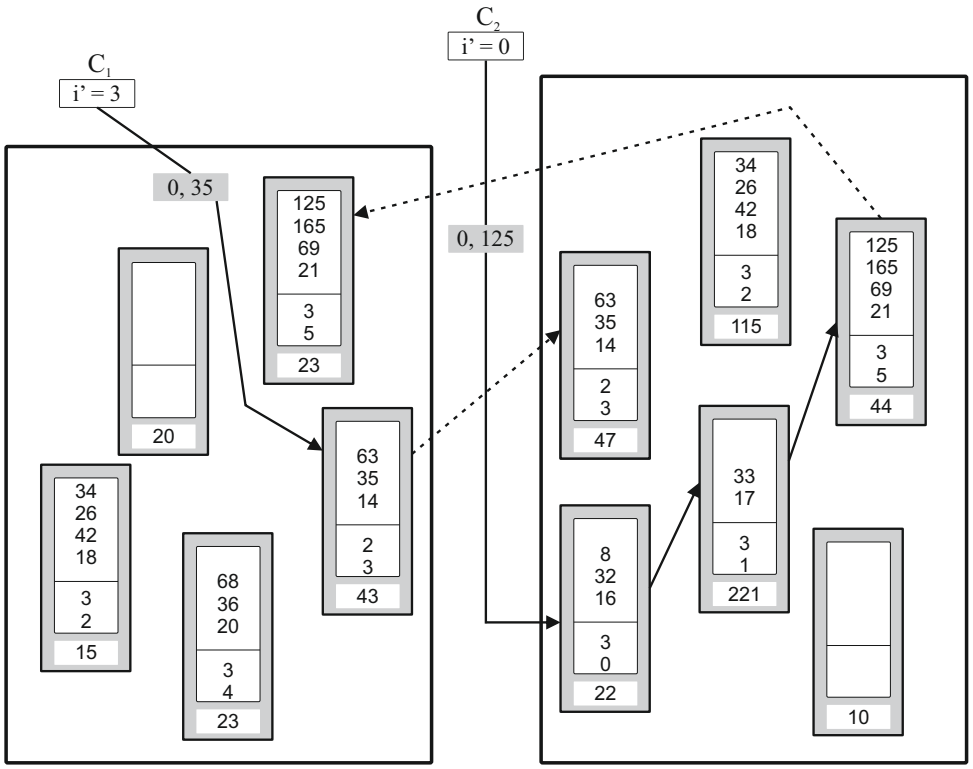


Obrázek 9: Základní schéma LH* se zrcadlením [3]

Kopie klíčů by měly být umístěny v různých bucketech při zachování dostupnosti přes hashovací funkci. Rozlišujeme dva typy zrcadlení na úrovni schématu LH*.

- *SA* - zrcadlení ($structurally - alike$), zde jsou jednotlivé parametry uzlu a bucketu shodné, a tedy i adresní výpočet je shodný.
- *SD* - zrcadlení ($structurally - dissimilar$) obsahuje rozdíly v adresování, a tedy i propagace změn může být pomalejší. Toto řešení může být výhodné na heterogenní síti, kdy nelze zajistit shodné podmínky přístupu.

SA-zrcadlení je jednodušší oproti SD, díky shodným parametrům souboru. Každý bucket má hlavičku, obsahující jeho úroveň i, a adresu $m = 0, 1, 2, \ldots$. Úroveň bucketu určuje naposledy použitá hashovací funkce při vkládání záznamů do bucketu.
Pokud by klient c_1 s úrovní $i' = 3$ vkládal klíč 35 do bucketu 3 souboru F_1 s úrovní $i = 2$ s uzlem 43. Podobně klient c_2 s úrovní $i' = 0$ posílá klíč 125 do bucketu 0 s úrovní $i = 2$. Projde toto vkládání souborem F_2 do správného bucketu a provede vložení jako v běžném režimu LH*. Koordinátor, který se na úpravě podílí, vyvolá na zrcadleném souboru F' vkládání, viz obrázek 10.
Veškeré úpravy jsou propagovány transparentně i pro klienty. Aby nedošlo k nekonzistenci, lze provádět změny až po ověření, že na zrcadleném souboru byla změna úspěšně provedena. To si samozřejmě vyžádá zprávy navíc.

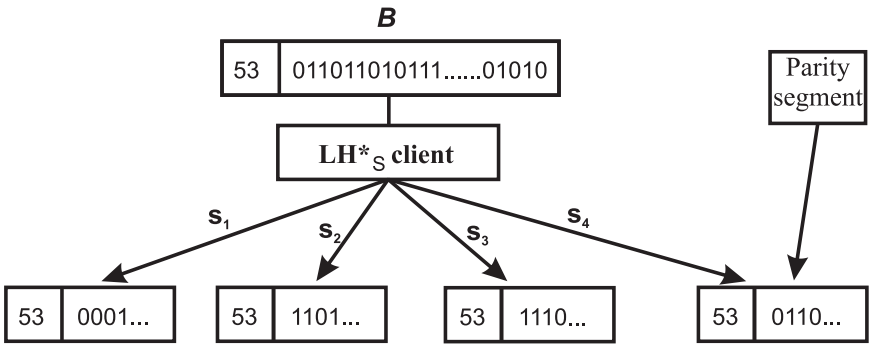


Obrázek 10: Ukázka vkládání při režimu zrcadlení [3]

3.2.3 LH*s

Záznamy v LH*s [4] jsou „proužkovány“ neboli segmentovány do $k > 1$ segmentů, vkládaných do rozdílných bucketů a uzlů. Součástí každého segmentu jsou i paritními bity pro obnovení segmentu mimo daný uzel. Toto schéma podporuje nedostupnost jednoho ze segmentů záznamu. Oproti typickému LH* režimu potřebuje na paritu LH*s

o 15 - 25% více datového prostoru.
Segmentování je založeno na nejnižší úrovni, jednotlivých bitech záznamu, vkládáním po sobě jdoucích bitů záznamu do rozdílných segmentů. Vytváří se tím ochrana proti proniknutí k datům v jakémkoliv místě a prozrazení jejich obsahu. V nejlepším případě totiž útočník získá celý segment obsahující 1 bit z k záznamů segmentů.
Vzniká zde určité zhoršení při vkládání záznamu, z důvodu zaslání parity segmentu. Vyhledání klíče k záznamu se může stát oproti LH* pomalejší, neboť klient obesílá k bucketů. Je zde potřeba více času CPU na zpracování dotazů a odpovědí, neboť je každý záznam poslán v nejméně 2 zprávách. Některé aplikace toto zhoršení akceptují pro jeho vysokou bezpečnost.



Obrázek 11: Rozptýlení záznamu do $k = 3$ segmentů [4]

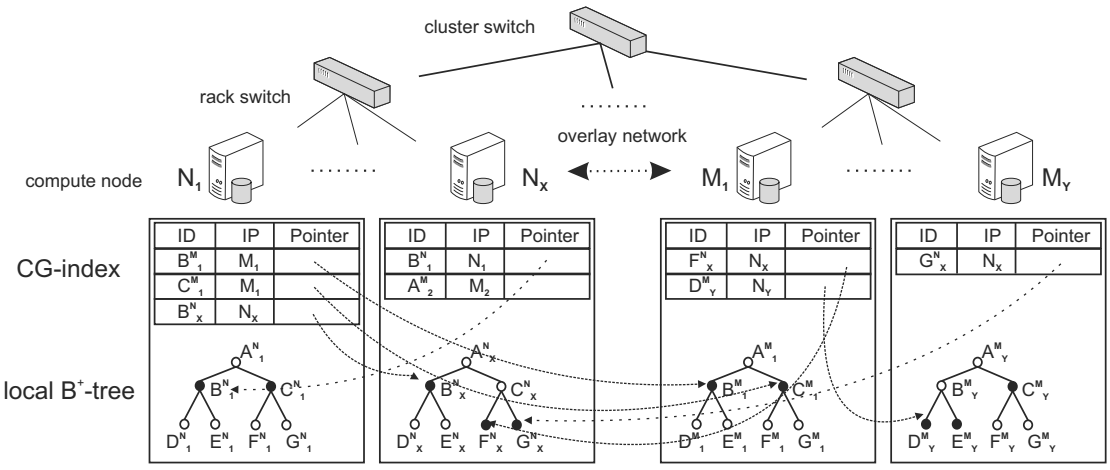
Záznam R má klíč c a sekvenci bitů B o velikosti mk, poslední bity mohou být doplněny nulou. Průběh vložení dat klientem je v LH*s následující (viz obrázek 11). Bit b'_j je paritní bit pro řetězec j-tého bitu každého segmentu $1 \leq j \leq k$. Pokud bude některý ze segmentů s z R nedostupný, paritní segment umožní rekonstrukci s.

3.2.4 LH*rs

LH*rs [5] se vyznačuje strukturováním škálovatelného souboru do skupin o m bucketech a využitím paritního kalkulu založeného na Reed-Solomon (RS) [5], mazacím opravném (de)kódování. Soubor obsahuje data záznamů a paritní záznamy.
Data jsou jako u LH* na začátku ukládána do bucketu b_0 na uzel A_0. Přetečení vyvolá rozdělení bucketu. Naopak mazáním a vyprázdněním bucketu vyvoláme sloučení bucketů. Sloučení vždy ovlivňuje naposledy rozdělený bucket v řadě. O oba procesy se stará koordinátor. Datová struktura záznamu a paritního záznamu je stejná jako u LH*g. Nejzajímavější je však způsob práce s paritními záznamy.
Dekódování paritního záznamu je v základu založeno na Erasure Correcting Codes (ECC), avšak je možné použít upravené Reed-Solomon (RS) kódování s použitím Galoisova pole (GF) [5]. Výpočet následně využívá existenci primitivních prvků v každém z GF a obsahuje několik robustních matic pro jednotlivé operace.

3.2.5 CG-index

CG-index [15] vytváří efektivní indexaci dat v prostředí cloudu za pomoci B$^+$-stromu. Hlavní myšlenkou je rozdělení záznamů na několik malých částí nazývaných střepy. Každý střep je distribuovaná jednotka uložená v unikátně vypočítaném uzlu. Místo budování celkového indexu uložených záznamů buduje CG-index pouze lokální indexaci zvanou index střepu. Tento index střepu je distribuční jednotkou CG-Indexu, která je uložena a spravována v unikátním indexu uzlu. Tím CG-Index dosahuje dané škálovatelnosti. Dotazy jsou provedeny výpočtem všech kompetentních střepů.
Na počátku vybudujeme lokální indexaci u každého uzlu, který si indexuje data v sobě uložená. Dále uzly organizujeme do překrývající se struktury a lokálně publikujeme části uzlů pro efektivní dotazování. CG-index podporuje běžné operace (insert, delete), ale hlavně vyhledání pomocí klíče a rozsahové dotazy (range query).

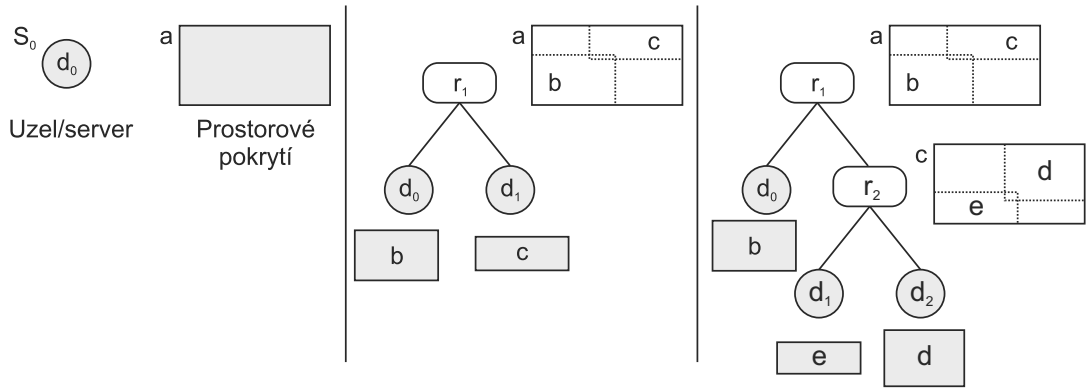


Obrázek 12: Architektura systému s distribuovaným B$^+$-stromem

Architektura komunikace je postavena na protokolu *BATON* [20] pro dynamické peer-to-peer spojení. Obrázek 12 ukazuje strukturu stanic v systému. Překrývající se směrovací protokol nám umožňuje vyhledávání od jakéhokoliv uzlu, nemusíme jej vždy započít u kořene. Varianta dotazování za pomoci broadcast zpráv by zde byla jednoduchá, avšak málo škálovatelná.
Budování globálního indexu (CG) na lokálních B$^+$-stromech je na obrázku 12 značeno černými uzly. Uzly si udržují *pointer* na CG-index, ten je tvořen klíčem B$^+$-stromu. Z komunikace si ukládáme jen data o publikovaných uzlech: *blk*, *range*, *keys*, *ip*. *Blk* je číslo datového bloku uzlu, *range* je rozsah daného uzlu, *keys* jsou vyhledávací klíče v uzlu a *ip* je IP adresa příslušného uzlu.

3.2.6 SDR-strom

Jedná se o SDDS strukturu využívající principy organizace R-stromu a splňující základní principy SDDS z definice 3.2. SDR-strom [8] je binární strom mapovaný z několika uzlů. Každý vnitřní uzel ukazuje právě na dva listy (směrovací uzel), hloubka stromu je rovna logaritmu počtu stromů.
Ukazatel na levý a pravý list obsahuje adresář obdélníků MBR, tedy levé a pravé podstromy. Každý listový nebo datový uzel obsahuje část indexovaných objektů. Server je jedinečně identifikovatelný podle ID ($ID = i$) a uložené dvojice (r_i, d_i) směrovacího a datového uzlu. Obrázek 13 ukazuje postupné vytváření SDR-stromu z uzlu S_0 na uzly S_1 a S_2. Při překročení kapacity uzlu nebo některých zvolených kritérií prostorového porytí



Obrázek 13: Vývoj SDR-stromu [8]

MBR dojde k rozdělení dat na d_0 a d_1. Vzniknou podprostory a = *mbr*(b ∪ c) MBR na uzlech a směrovací uzel r_1 mezi nimi. Směrovací uzel obsahuje své ID a odkazy na potomky. Odkaz na potomka obsahuje informaci ve formě čtveřice parametrů (ID potomka, seznam MBR, hloubku podstromu, typ uzlu). Typ uzlu může být datový, na něm se data z adresáře MBR přímo nacházejí, nebo směrovací na další uzly/listy.
Pohled klientů na strukturu distribuovaného stromu je řešen **IAM** (image adjustment message) [1] zprávami. Klienti se mohou dopustit adresní chyby při dotazu na uzel DDS (server). IAM zajišťuje správnou adresaci dotazu klientem. V každé odpovědi uzlu je obsažena čtveřice odkazů tvořená z datového a směrovacího uzlu a odkazů na levého a pravého potomka.

4 Návrh konceptu SDDS

Pokoušíme se zde shrnout a využít jednotlivé vlastnosti a poznatky z předešlých kapitol k navržení vhodné komunikace, rozložení dat a pokusit se dodržet co možná nejvíce kritérií definujících SDDS struktury.

4.1 Omezení a požadavky

Vlastní implementace některého konceptu LH* nepřipadá v úvahu s ohledem na složitost a časovou náročnost implementace. Zmíněné koncepty LH* struktur jsou spojené i s několikaletou výzkumnou prací podobně jako stromové DS. V práci byl použit rámec QuickDB [24] implementovaný výzkumnou skupinou Database Research Group[1] na Katedře informatiky[2]. Poskytnutím již implementovaných datových struktur - databází, viz kapitola 4.2, se lze věnovat konceptu komunikace a způsobu rozložení dat mezi uzly.
Pro udržování pohledu nad rozdělením vyjdeme ze systému *Range parttitioning* [28] založeného na rozdělení rozsahu hodnot klíčů mezi jednotlivé uzly. Zde se i přes statický režim dělení klíčů pokusíme přiblížit dynamičnosti za pomocí rozšíření klíčů na více uzlů. Podobně jako u SDR-stromu použijeme tvorbu lokálního (až globálního) pohledu na uzly klienty.
Návrh konceptu komunikace mezi jednotlivými aplikacemi, klienty a servery je zásadní. Komunikace využije stávající protokoly sítě *TCP/IP*. Vlastní tvorba transportní vrstvy k nahrazení TCP/IP a UDP protokolů by byla značně složitá a časově nezvládnutelná. Pro implementaci byl zvolen *výhradně jazyk C++* [13]. V implementaci se snažíme omezit počet přístupů na disk, jelikož je to časově velmi náročná operace. Alokujeme si proto předem větší prostor v operační paměti. Využitím ukazatelů na objekty a jejich předáváním mezi metodami se vyhneme zbytečné duplikaci proměnných. Za chodu aplikace není příhodné dynamicky alokovat paměť, jelikož alokace a uvolňování paměti jsou operace srovnatelné s diskovým přístupem. Je nutná alokace paměti před operacemi s datovou strukturou, proto je použit jako programovací jazyk právě C++ s operátory `new` a `delete`. Alokace a uvolňování paměti neprobíhá soustavně podle potřeb z důvodu časové náročnosti.

4.2 QuickDB

Poskytnuté datové struktury R-stromu a B-stromu jsou zprostředkovány za pomoci třídy `cQuickDB`. Třída vytváří pro datové struktury perzistentní datové prostředí, viz kapitola 4.7.
Každý záznam je tvořen klíčem a daty záznamu. Klíč může být tvořen různými datovými typy. Je definován dynamicky za pomocí tříd dědících z `cDTDDescriptor`. Dědičnost zajišťuje různorodost klíčů podle potřeby DS. V našem případě byla použita třída `cSpaceDescriptor` definující n-tici klíčů tvořící klíč typu `cTuple`.
V ukázce 1 je vytvořen popis pro n-tici klíče o délce = KEY_DIMENSION. Na popisu klíče

[1] http://db.cs.vsb.cz
[2] http://www.cs.vsb.cz

lze vytvořit objekt klíče, v ukázce 1 je reprezentován proměnnou `tp`. Objekt `cQuickDB` potřebuje při své inicializaci nastavení velikosti perzistentní datové struktury. Po vytvoření souborů na disku a alokaci paměti RAM vytváří DS B-stromu. Vstupní parametry konstruktorů jsou popsány v ukázce 1, níže.

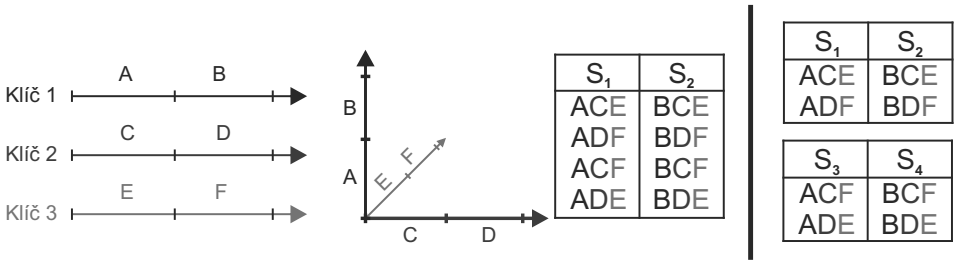
```
1   sd_BTREE = new cSpaceDescriptor(KEY_DIMENSION, new cUInt());
2   // uzel
3   cTuple *tp = cTuple tp(sd_BTREE);
4   // cQuickDB
5   quickDB = new cQuickDB();
6
7   if (!quickDB->Create(DB_Name, //název souborů DS
8            CACHE_SIZE, //počet uzlů v paměti RAM
9            MAX_NODE_INMEM_SIZE, //počet uzlů
10           BLOCK_SIZE) //velikot diskového bloku
11          ){return false;}
12  // B-tree hlavička
13  cBpTreeHeader<tKey> *mHeader = new cBpTreeHeader<tKey>(
14          "Btree1",// název DS
15          BLOCK_SIZE, //velikot diskového bloku
16          sd_BTREE,// cQuickDB
17          tp.GetSize(sd_BTREE),//velikost klíče zaznamů
18          DB_DATA_LENGHT*sizeof(char),//délka záznamu
19          false); //variabilní délka
20  // B-tree
21  cBpTree<tKey> *mIndexBtree = new cBpTree<tKey>();
22  // B-tree tvorba
23  if (!mIndexBtree->Create(mHeader, quickDB)){return false;}
24  cout<<"NEW Database BTREE was created!"<<endl;
```

Výpis 1: Ukázka vytvoření objektu QuickDB a B-stromu

Na třídy popisující klíč a DS jsme museli navázat. Použít je jako základ pro definici pohledu a vlastnosti DS. Tím jsme zachovali dynamičnost implementace - *nezávislost na typu klíče či použité DS*.
Třída `cMBRectangle` je využita pro popis pohledu na udržované záznamy DS. Neudržujeme pomocí něj pouze seznam rozsahu klíčů, ale využíváme i některé metody pro zjištění:

- `IsInRectangle()` - je tento klíč nebo MBR obsažen v MBR.
- `IsIntersected()` - je tento MBR protnut nebo obsažen v MBR.

Pomocí těchto metod jsme schopni rozhodovat na uzlu, zda daný příkaz s klíčem patří uzlu nebo má dojít k jeho přeposlání na jiný uzel.
MBR je u R-stromu definován dvěma body Q_{low} a Q_{high} ve vícerozměrném prostoru. B-strom není vícerozměrná DS, přesto zde využijeme k popisu intervalu klíčů třídu `cMBRectangle`, viz obrázek 14.



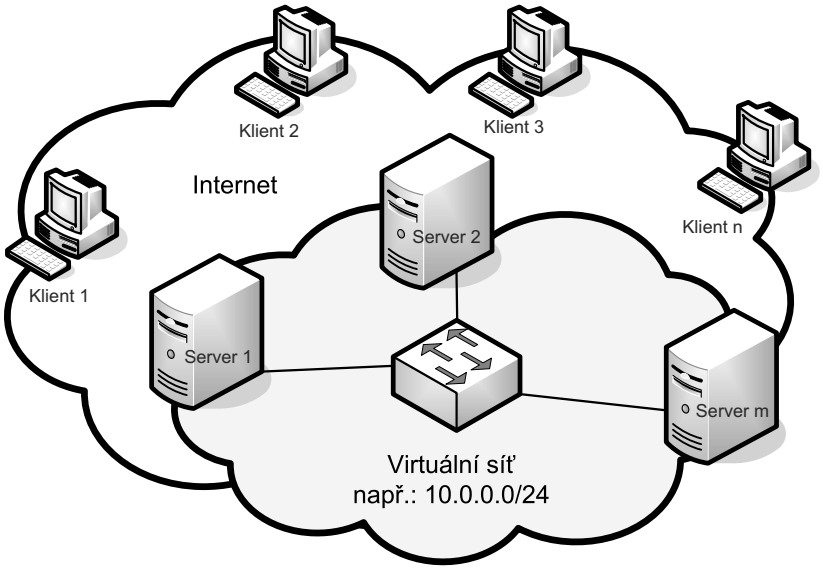
Obrázek 14: Rozdělení rozsahů klíčů

4.3 Návrh vlastní DDS

Koncept se musí vyhnout jakémukoliv centrálnímu zařízení a řízení v síti. Pokud navrhneme hvězdicovou strukturu serverů, pak se klient na počátku dotáže přes středový server, který by informace neustále přeposílal. Tento server by vlastnil tabulku adres serverů s jejich obsahem. Obsahem zde míníme například interval uloženého B-stromu, nebo jiná klíčová data. Nejenže by se stal přetíženým, ale při výpadku tohoto serveru by došlo k nedostupnosti všech serverů.
Řešením je vytvoření tabulky serverů na všech serverech. Poté by se stal každý server nezávislý. Klienti by se při první komunikaci mohli ve zpětné odpovědi dozvědět záznam, či dokonce záznamy o ostatních serverech. Musíme však zajistit řešení několika problémů.
Udržovat jednotlivé záznamy serverů aktuální pomocí rozeslání zprávy všem serverům není při velkém počtu serverů nejšťastnější, avšak nezavrhujeme jej pro náš koncept. Navíc pokud chceme být schopni rychle reagovat na případný výpadek, *nevyhneme se periodickému zasílání zpráv typu "HELLO"servery.* Navržený koncept komunikace se pokouší řešit veškeré výše uvedené problémy.

4.4 Síťové prostředí

Dnešní technologie TCP/IP dovolují přes internet vytvářet virtuální sítě, tunely, zapouzdřit komunikaci do různých protokolů a mnoho dalších možností. Vyhneme se složitému směrování a možným heterogenním částem sítí v Intranetu či Internetu.
S využitím různých technologií lze vytvořit virtuální síť přes Internet, na níž by byly umístěny naše servery. Topologie jakékoliv sítě by tak vypadala jako na obrázku 15. Virtuální středový přepínač by ve skutečnosti mohl být tvořen i několika směrovači s ústími tunelů od serverů. Tím by výpadek směrovače neohrozil celou virtuální síť. Za pomoci nakonfigurovaných pravidel ve směrovacích tabulkách směrovačů a s jejich dnešní rychlostí konvergence by se výpadek pokusili okamžitě nahradit zbylé směrovače mezi sebou. Konfiguraci a použité technologie zde neuvádíme, neboť jsou závislé na samotných zařízeních a kritériích přenosové sítě.
Za pomoci Ethernet protokolu podporujícího zasílání zpráv broadcast a multicast po segmentu sítě, lze obeslat jednotlivé prvky-servery. Čas průchodu zprávy sítí je stejný, jako u Point-to-point zpráv, avšak musíme zabránit „záplavě“ příliš mnoha míst. Odběr



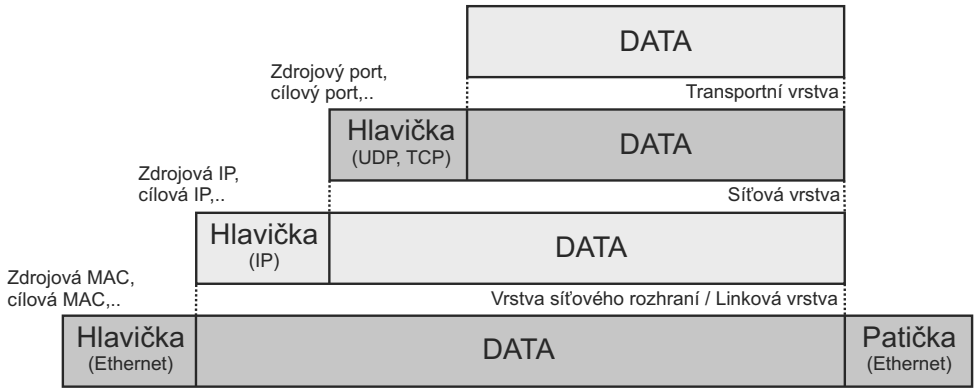
Obrázek 15: Vzhled virtuální sítě

multicastu by se u serverů mohl využít v budoucnu pro oznamování uzamknutí určitého záznamu či skupiny záznamů. Pro přenos (replikaci) záznamů, však není vhodným řešením, viz kapitola 4.4.1.

4.4.1 Komunikace na síti

Pro komunikaci a přenos dat lze použít dva protokoly z rodiny TCP/IP na transportní vrstvě. Prvním je spojový protokol TCP. Hlavní výhodou je spolehlivý přenos dat. Využije se při zasílání příkazů k operacím nad databází a samotnými záznamy. Jsme schopni okamžitě informovat o nedostupnosti cíle či výpadku spojení a reagovat na ně. Nevýhodou zůstává delší časová náročnost, vyplývající z nutnosti navázání a ukončení spojení oběma stranami.
Druhý protokol UDP obsahuje hned v prvním rámci aplikační data, je tedy méně náročný. Ušetříme šířku pásma a čas strávený nad sestavením spojení. Vše bohužel za cenu neschopnosti zaručit doručení do cíle. Využití nachází tento protokol v periodickém oznamování přítomnosti serveru na médiu/síti a šíření méně prioritních informací.
Kde jaký protokol využijeme, si musíme dobře promyslet. Nelze dopustit, aby se pohled na data stal neintegritní, či dokonce nekonzistentní. Některé prvky sítě (směrovače, přepínače) si mohou při kritickém zaplnění své cache paměti odlehčit zahozením některých rámců a to jak u UDP, tak TCP komunikace. Jak je vidět na obrázku 16, naše zaslaná data jsou zapouzdřena postupně do rámců. Velikost pro naše data určuje použitá transportní vrstva TCP nebo UDP a následně hodnota MTU rámce. Hodnotu MTU neměníme z důvodu složitosti nastavení klienta a serveru, ale možnosti nepřijetí dohodnuté velikosti kterýmkoliv síťovým prvkem na trase komunikace.
U TCP se v SYN zprávě při navázání spojení uvádí velikost celé zprávy a navržena ve-

likost MTU rámce TCP. Celý TCP rámec má standardně maximální velikost 1 500 bytů (nepoužíváme-li Jumbo Ethernet 8 kB), samotná hlavička má standardně 40 bytů (u RFC 1 323 přidáváme 12 extra bytů k hlavičkce). Na data nám v jedné zprávě TCP maximálně zbývá pouze 1 460 bytů. Při testech, viz kapitola 6.3.6, měl jeden aplikační datový rámec pro data 1 260 B. Menší velikost než 1 460 bytů je způsobena zapouzdřením hlaviček dalších vrstev (aplikační, prezenční a relační).



Obrázek 16: Zapouzdření dat v TCP/IP [16]

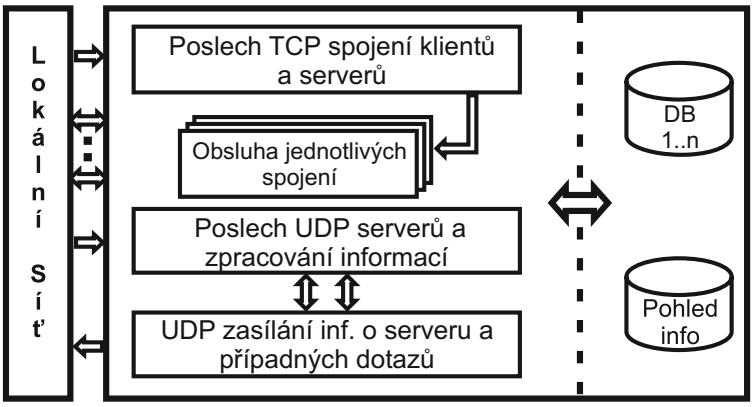
Pro rozlišení jaká data/typ jsou vlastně obsažena, je nutné v hlavičce rámce uvést hodnotu určující typ, a tedy obsah zprávy. Podle něj se zbývající byty zapíší do jednotlivých proměnných. Problém může nastat, pokud chceme zaslat více objemných záznamů, například z rozsahového dotazu. Data se zpracují na serveru a pošlou klientovi odpověď po více rámcích. Otázkou zůstává, jak velký dotazový rozsah povolit a ošetřit tak maximální velikost zasílaných dat na síti.

4.4.2 Server

Jednotlivé činnosti/procesy aplikace serveru jsou nastíněny na obrázku 17. Implementovaný server musí zvládat vícevláknovou obsluhu přijatých spojení jak na TCP, tak UDP socketu, viz kapitola 6.4.4. Vlákna na serveru se dají rozdělit do dvou skupin podle použitého komunikačního protokolu.
Vlákno pro poslech socketu TCP na IP adrese a portu. Při zachycení komunikace následuje okamžité předání spojení (socket navázaného spojení) vláknu k vyřízení. Jedno či více vláken pro vyřízení navázaných spojení TCP. Druhou skupinu tvoří vlákno pro poslech socketu UDP na IP adrese a portu. Poslech broadcast zpráv a jejich následné vyřízení (zpracování zpráv typu *"HELLO"* apod.). Vlákno pro cyklické zasílání informací o serveru, popřípadě reakcí na přijatou zprávu broadcast.

4.5 Chování SDDS

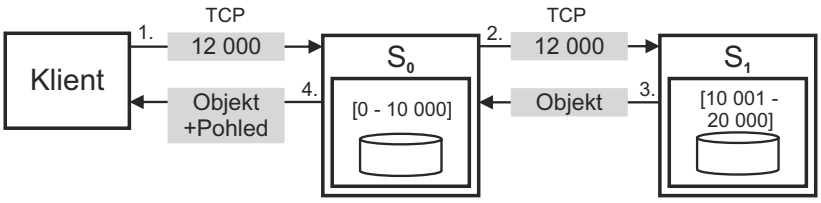
Popisujeme jednotlivé případy chování aplikace klienta a serveru (uzlu DDS) při komunikaci a případné chování v rozporu s definicí SDDS.



Obrázek 17: Jednotlivé činnosti/procesy serveru

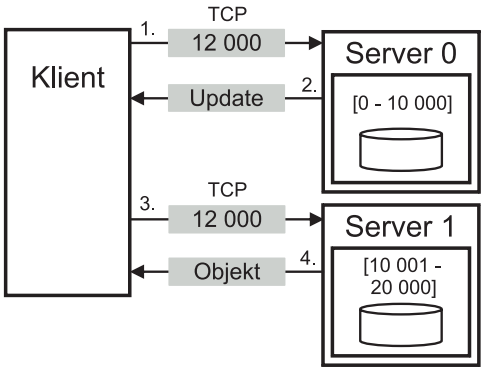
4.5.1 Bodový dotaz

Klient se chce dotázat na záznam s klíčem $a = 12\,000$. Pohled klienta může, ale nemusí obsahovat adresu serveru uchovávající daný záznam. Pokud klient zná tento server, pak proběhne dotaz a daný záznam je navrácen.
Zajímavější situace nastane v případě, kdy server s klíčem a neznáme, viz obrázek 18. Klient se vědomě dopustí adresní chyby a dotáže se na S_0. Server zjistí adresní chybu klienta a dotaz přepošle na správný server s příznakem přeposlání ve zprávě. Server S_1 dotaz zpracuje a přidá o sobě informace (IAM) pro klienta. S_0 jako prostředník pouze zprávu přepošle a dále nezpracovává. Klient získá se záznamem i rozšíření pohledu o další server a při dalším dotazu na klíče z S_1 se adresní chyby nedopustí.



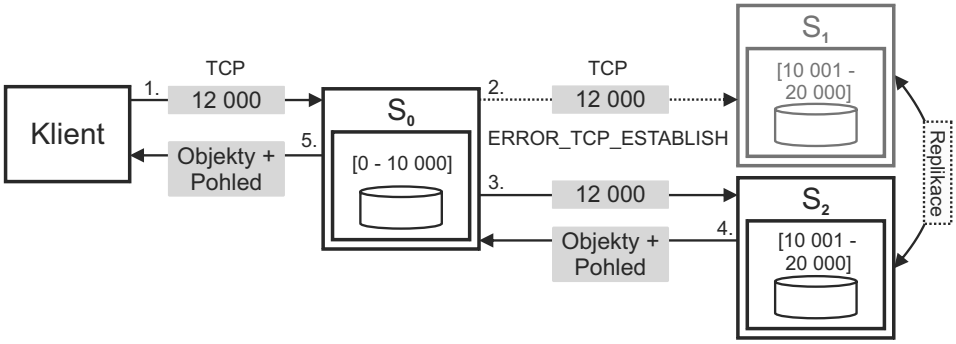
Obrázek 18: Vyhledání s adresní chybou

Nevhodné provedení bodového dotazu je zobrazeno na obrázku 19. S_0 musí sestavit *Update* pohledu pro klienta na S_1. Pokud S_0 drží v paměti dotaz i informaci pro přeposlání je nevhodné tuto situaci nevyužít. Zašleme *Update* pohledu, který si klient zpracuje a následně znovu naváže spojení se serverem. Nejen, že jsou všechny procesy (2x navázání spojení, práce s pohledem) časově náročné, ale jsou i proti definici SDDS. Server se může stát nedostupným z důvodu výpadku sítě nebo z důvodu údržby. Pak je vhodné aplikovat replikaci záznamů na jiný server. Předešlý průběh dotazu obohacený o nedostupný server je zachycen na obrázku 20.
Pro zajištění integrity záznamů a celé databáze by se po zprovoznění serveru S_1 měl



Obrázek 19: Ukázka špatného konceptu pro vyhledání s adresní chybou

synchronizovat s S_2. Vhodným řešením by byl log soubor na S_2 do něhož by byli zaznamenány úpravy, které na S_1 nebylo možno replikovat v době nedostupnosti. Naše řešení v případě opakované nedostupnosti serveru příkaz neprovede.



Obrázek 20: Průběh bodového dotazu s nedostupným serverem

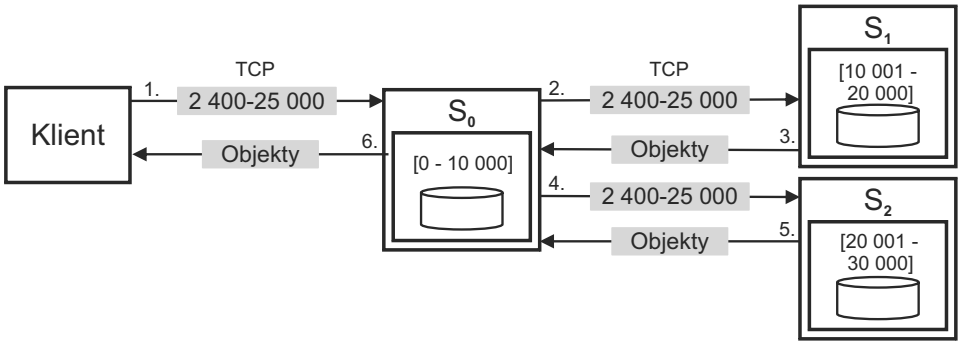
Poznámka 4.1 Aplikace schopnost synchronizace serverů pomocí log souboru při replikaci dat neimplementuje. Nelze tudíž zaručit konzistenci DB při výpadku jednoho ze serverů.

Poznámka 4.2 Reakce na nedostupnost jednoho ze serverů je dána časovou náročností zjištění nedostupnosti. Je vhodné uvažovat o cyklické aktualizaci pohledu na serveru podle časových razítek posledních zpráv „HELLOU“, viz kapitola 4.5.3.

4.5.2 Rozsahový dotaz

Vyhledání záznamů pomocí rozsahového dotazu s klíči Q_{low} = 2 400 a Q_{high} = 25 000 je zachyceno na obrázku 21. Klient se i zde může dopustit adresní chyby a dotázat se na server, který daný rozsah klíčů neuchovává. Stane se z něj podobně jako v případě

bodového dotazu pouhý prostředník celé transakce. Server S_0 drží dolní rozsah, a tak provede první dotaz na své DB. Následně server S_0 vyhledá seznamy serverů držících zbylé klíče. Vyhledání ostatních serverů provede jen v případě, že sám neuchovává rozsah klíčů a_{low} až a_{high}. Pro získání ostatních výsledků přidá S_0 k dotazu klienta příznak přeposlání. Díky příznaku přeposlání server S_1 a S_2 přijatý dotaz okamžitě provedou nad databází a výsledné záznamy vrátí serveru S_0. Získané výsledky dotazu se na S_0 spojí do jedné zprávy pro klienta.



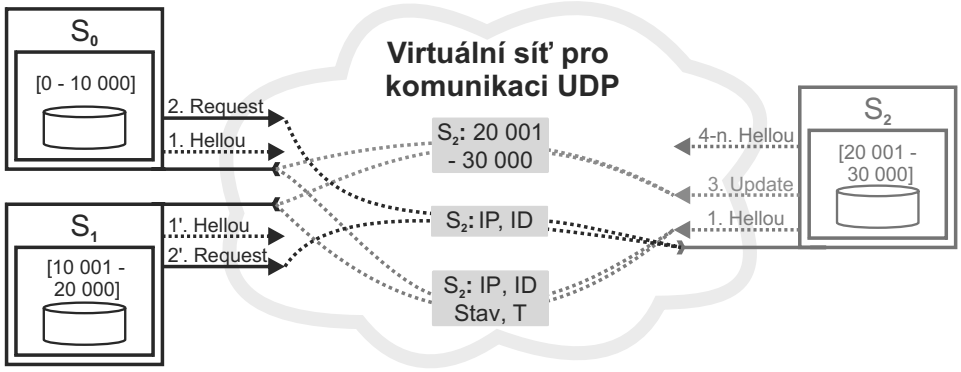
Obrázek 21: Průběh rozsahového dotazu

Poznámka 4.3 Všimněme si, že se S_0 stane pro dotaz centrálním prvkem koordinujícím celý dotaz podobně jako v prostředí MapReduce, avšak bez nutnosti kontroly redundance vrácených hodnot.

4.5.3 UDP zprávy

UDP komunikaci zajišťují dvě vlákna sdílející informace o přijatých zprávách a pohledu serveru na okolí. Mechanizmus *"Hellou"* zpráv zajišťuje škálovatelnost počtu serverů za běhu společně s vyvažováním zátěže. Vyvážení serverů zajišťuje parametr zátěže **T** a zasílání zpráv periodicky v určeném časovém intervalu. Ukázka průběhu UDP komunikace je na obrázku 22. Server zpracuje zachycenou UDP komunikaci a následně na ni reaguje ve druhém vlákně zasláním jedné z následujících zpráv:

- **Hellou** - průběžná zpráva oznamující pouze ID, IP adresu, stav a zátěž T serveru, který ji zaslal.
- **Request** - reakce na zachycení zprávy *Hellou* od dosud neznámého serveru (není uveden v pohledu serveru).
- **Update** - reakce na zachycení zprávy *Request* se shodnou ID a IP adresou naslouchajícího serveru. Zpráva obsahuje celé nastavení serveru - $ServerConfig$. Jedná se o IAM zprávu.
- **Close** - zpráva oznamující ID a IP adresu serveru, který byl vypnut.



Obrázek 22: UDP komunikace mezi servery s přidáním serveru S_2

Nedostatkem implementace a předloženého konceptu je hromadná reakce všech serverů na nový server S_2 zprávami *Request*. Po dobu jedné periody nedojde k zaslání zprávy *Hellou* všemi servery, a tedy k možnosti vyvažování zátěže. Otázkou je, jak často přidáváme nový server.

Poznámka 4.4 Periodu zasílání UDP zpráv serverem lze nastavit v `cServerDefinition.h` parametrem *TIME_SEND_HELLOUS*, použitá hodnota u serverů byla vždy 1s.

4.6 Vyvážení zátěže

Vyvážení zátěže -„Load balancing“ [4] mezi servery. Kapitola 4.5.3 nastínila průběh UDP komunikace zajišťující zprávy *Hellou* potřebné pro následující dva druhy vyvážení:

- Vyvážení rozdělení dat mezi servery
- Vyvážení dotazů mezi servery (nutná replikace dat)

Dynamická tvorba, dělení a mazání pohledu na uložení dat mezi servery byla nahrazena statickou definicí rozsahů/intervalů na jednotlivých serverech. V rámci masivně paralelního zpracování dat je pro nás zajímavější se věnovat spíše vyvážení dotazů mezi servery, než dynamickému rozdělení rozsahu/intervalu klíčů mezi uzly DDS. Vyvážení dotazů bez centrálního koordinátora je dosti složité. Nelze odhadnout nenadálé zatížení klienty některého ze serverů. Řešení, kdy server začne zamítat spojení od klientů je nepřípustné. Nelze vědět, zda klient vůbec zná jiný server pro dotaz. Opakování dotazu na server je proti definici chování SDDS. Nabízí se nám následující scénář pro řízení zátěže:

1. Z UDP komunikace získávat periodicky přehled o zátěži okolních serverů.
2. Při dotazu detekovat přetížení serveru oproti ostatním se stejným obsahem.
3. Upozornit klienty při přetížení na využití jiných serverů pro příští dotazování.

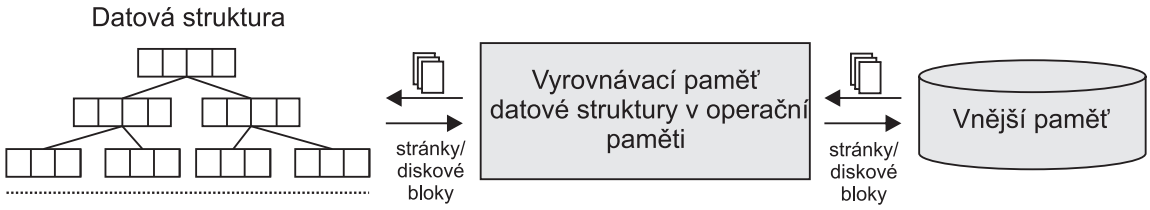
4. Nastavit u klienta vhodné chování při výběru serveru pro vykonání dotazu a zpracování odpovědi.

Za určitou dobu předpokládáme částečné ustálení zátěže rozdělením serverů mezi klienty. Za možný problém pro adekvátní testy a nastavení se jeví simulace klientů. Zda simulovat skupinu klientů, která se časem naučí pohled na servery a zůstane tak po zbytek testu. Nebo po určené době resetovat pohledy klientů a simulovat nové klienty s adresními chybami.

Poznámka 4.5 Vyvážení zátěže dotazů mezi servery se musí vyhnout hromadnému „přelévání " klientů z jednoho serveru na druhý a musí být pokud možno postupné [4].

4.7 Použité datové struktury

Implementace datové struktury R-stromu a B-stromu byly k testování poskytnuty, viz 4.2. Inicializační nastavení obou struktur požaduje zadání počtu dimenzí/prostorů a velikost indexovaných dat. Tyto hodnoty se mění podle indexované databáze. Obě struktury



Obrázek 23: Perzistentní datová struktura

jsou perzistentní, viz obrázek 23, tedy využívají vyrovnávací paměť v hlavní operační paměti k dosažení větší rychlosti získání záznamu. V průběhu testování byly parametry nastaveny následovně:

Parametry	**B-strom**	**R-strom**
$BLOCK\_SIZE$ (velikost diskového bloku) [B]	2 048 B	2 048 B
$CACHE\_SIZE$ (počet uzlů v op. paměti)	195 000 - N	195 000 - N
$MAX\_NODE\_INMEM\_SIZE$	2 400	2 400

Tabulka 1: Nastavení DS

$BLOCK\_SIZE$ parametr je dán velikostí diskového bloku a nabývá násobků hodnot: 2 048. Další parametr $MAX\_NODE\_INMEM\_SIZE$ je na předešlém závislý. Měl by být nastaven přibližně na $BLOCK\_SIZE * 1.1$, tedy o 10% větší hodnotu. Parametrem $CACHE\_SIZE$ ovlivníme počet uzlů udržovaných v operační paměti. Jsme pouze závislí na dostupné paměti stroje, proto uvádíme v tabulce - N.

Poznámka 4.6 DS si při nastavení podle tabulky 1 alokuje v operační paměti virtuálních strojů cca 500 - 550 MB.

5 Návrh komunikace DDS

V předešlé kapitole jsme si uvedli, že budeme využívat protokoly TCP a UDP pro komunikaci na transportní vrstvě. Nyní popíšeme vlastní rámec a způsob serializace dat na aplikační vrstvě. Průběh implementace dal postupně vzniknout několika třídám serializujícím komunikaci mezi klientem a serverem.

5.1 První návrh

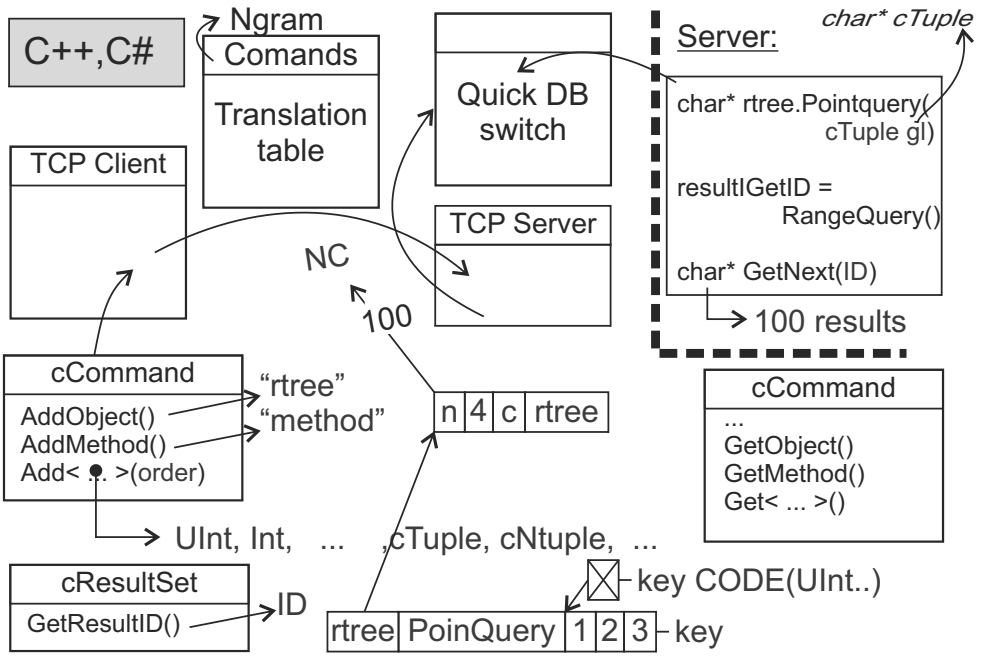
První komunikace počítala s několika druhy zpráv s rozdílným obsahem. Vznikla třída `myPacket`, v níž se podle první hodnoty `PType` v záhlaví zprávy načítal další obsah. Postupně vzniklo 15 druhů rámců pro jednotlivé operace. Přijatý řetězec znaků zprávy se předal metodě `Read()`. První bajt řetězce signalizoval druh použitého rámce, následně se pomocí příkazu `switch(PType)` aplikovalo čtení hodnot ve zbytku zprávy do parametrů objektu `myPacket`. Přes proměnné objektu `myPacket` se dále interpretoval obsah zprávy.
Statická definice celého obsahu s datovými typy však není vhodná pro heterogenní nároky databází. Neustálé přepisování a úpravy obsahu rámců by byly neúnosné. Z tohoto důvodu se od této komunikace upustilo. Nelze však přehlédnout programovou jednoduchost a malou náročnost na serializaci zprávy při přijetí/zaslání.

5.2 Komunikace DDS v2.0

Zhodnocení nedostatků první implementace a jazyka C++ [13] dalo vzniknout vlastnímu návrhu vzdálené komunikace za pomocí dvou rámců pro přenos dat. Na počátku stál náčrt konceptu komunikace, viz obrázek 24. První typ rámce tzv. *Command* zajišťuje volání určité metody a přenos vstupních parametrů. Zpětnou vazbu zajišťuje druhý rámec pro přenos výsledku volané metody, tzv. *ResultSet*. Oba rámce dědí z abstraktní třídy `cFrame`, to zajišťuje kompatibilitu při (de)serializaci přenosu metodami třídy `cSocket` a možnost tvorby dalších rámců.

Definice 5.1 ***RPC*** *- vzdálené volání metod, umožňuje aplikaci jednoduché volání metod objektů druhé aplikace obvykle běžící na jiném počítači.*

Před návrhem a implementací byly zkoumány možnosti jazyka C++ volat vzdáleně metody objektů. Předpokládali jsme integraci nástrojů pro model RPC v nové platformě nástrojů pro Visual Studio 2011. Hledání a pokusy v novém vývojovém prostředí bohužel nenaznačily implementaci potřebných metod, a tedy ani integraci modelu RPC. Hledání následně pokračovalo v dostupných knihovnách API pro C++ (např. *ICE* [23]). Po prozkoumání zdrojových kódů se ukázalo, že se žádné dynamické volání ve smyslu definice RPC neprovádí. Třídy využívaly předpřipravený interface s třídou jako prostředníkem definujícím datové typy a popis metod pro *READ/WRITE* jednotlivých parametrů.



Obrázek 24: Návrh komunikace klient/server

5.2.1 Serializace

Pro přenos hodnot datových typů a objektů vznikla třída `cCode` serializující data do struktury `Parametr`, viz výpis 2. Definujeme (de)serializaci základních datových typů na `Parametr`. U objektů potřebujeme, aby třída implementovala (de)serializaci v metodách `Read()` a `Write()` do řetězce typu `char`. Dále jen stačí definovat v `cCode` hodnotu, o jaký datový typ či třídu se jedná. Nevýhodou je nutnost definice a dopsání požadovaného volání metod `Read()` a `Write()` pro daný objekt v `cCode`. Jednodušší či elegantnější řešení jsme nenalezli.

```
1 //struct for reprezentation parametrs/objects
2  typedef struct Parametr{
3    unsigned char code;
4    unsigned int size;
5    void *ptr;
6  }P;
```

Výpis 2: Struktura udržující de/serializované objekty a datové typy

Serializace byla testována na všech základních datových typech v C++ a objektech poskytnutých s DS, viz kapitola 4.2. Při tvorbě klienta v jazyce C# byl otestován přenos částí textu bakalářských a diplomových prací na server bez ztráty či nekonzistence diakritiky. Řetězec textu byl serializován přes datový typ `wchar_t`.

5.2.2 Command

Volání metod (příkazů) se serializuje za pomoci třídy `cCommand` obsahující definici rámce pro přenos parametrů, viz obrázek 25. Pro správné použití příkazu na obou stranách musí být definované hodnoty pro jednotlivé objekty a metody shodné, v našem případě pro objekty představující DS na serveru. Hlavička rámce obsahuje hodnoty definující:

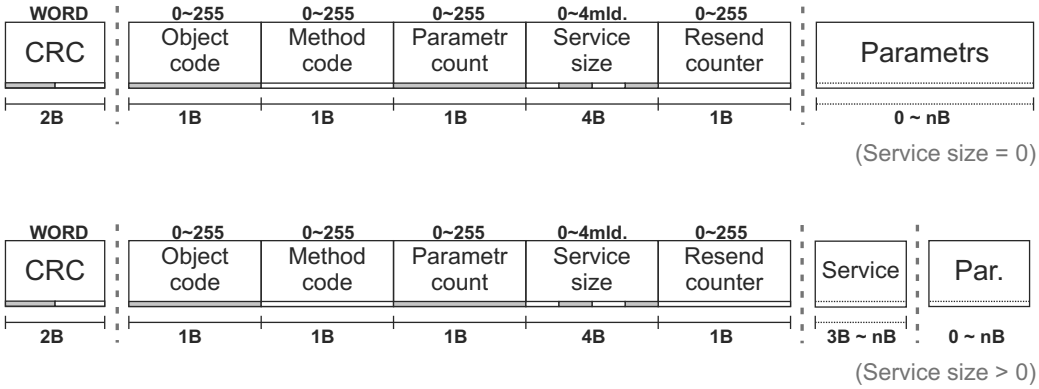
- **CRC** - hodnota kontrolního redundantního součtu rámce počítaná a kontrolovaná při zasílání a příjmu rámce. Pokud CRC nesouhlasí, pak byla data porušena při přenosu nebo nebyla přijata celá zpráva.
- **Object code** - identifikátor volaného objektu. Slouží k odlišení typu volaného objektu. Například pro rozeznání objektu B-stromu, R-stromu apod.
- **Method code** - identifikátor volané metody objektu. Na serveru slouží k odlišení metod objektů, například: Insert, PointQuery, Close atd.
- **Parametr count** - počet přenášených parametrů, které jsou uloženy ve zprávě, v poli **Parametrs**. Napomáhá blíže určit volanou metodu v případě přetížené metody.
- **Service size** - velikost přenášené servisní informace uložené v části *Service*.
- **Resend counter** - počítadlo přeposlání příkazu mezi servery, slouží k zabránění zacyklení příkazu a rozeznání, zda příkaz přišel od klienta nebo jiného serveru.
- **Service** - pokud je hodnota $Service\ size > 0$, pak je zde obsažena servisní informace ve formě serializovaného objektu $cService$.
- **Parametrs** - pokud je hodnota $Parametr\ count > 0$, pak jsou zde obsaženy parametry metody.

Samotný objekt lze zapsat jako parametr a vložit do jiného objektu `cCommand` jako parametr.

5.2.3 Result Set

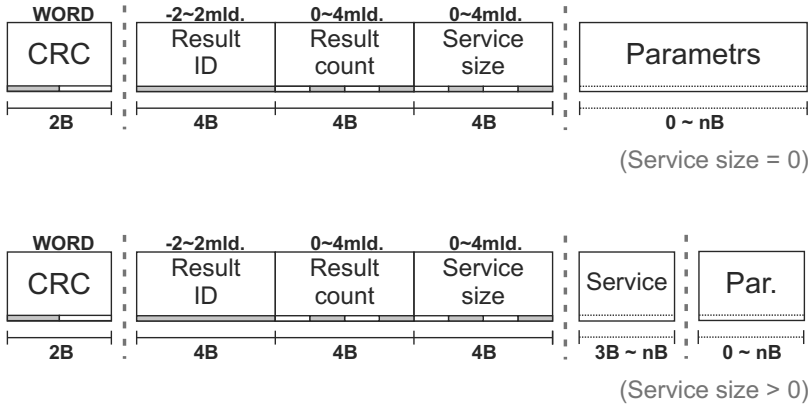
Navrácení výsledku metody obstarává třída `cResultSet`. Obsahuje definici rámce pro přenos návratových hodnot, viz obrázek 26. Jednotlivé hodnoty jsou stejně jako u příkazu serializovány do struktury `Parametr`. Hlavička rámce obsahuje tyto hodnoty:

- **CRC** - hodnota kontrolního redundantního součtu rámce počítaná a kontrolovaná při zasílání a příjmu.
- **Result ID** - > 0, jednoznačný identifikátor výsledku na serveru použitelný pro volání vrácení zbytku výsledků.
  - $= 0$, volání metody proběhlo úspěšně a všechny výsledky jsou obsaženy v poli parametrů.



Obrázek 25: Struktura rámce Command s/bez servisní informace

  - < 0, nastala chyba při volání metody a je zde navrácena hodnota chyby.
- **Result count** - počet přenášených položek uložených v části $Parametr$.
- **Service size** - velikost přenášené servisní informace.
- **Service** - pokud je hodnota $Service\ size > 0$, pak je zde obsažena servisní informace ve formě serializovaného objektu $cService$.
- **Parametrs** - pokud je hodnota $Parametr\ count > 0$, pak jsou zde obsaženy položky výsledku.



Obrázek 26: Struktura rámce ResultSet s/bez servisní informace

Poznámka 5.1 Objekty `cCommand` i `cResultSet` lze zapsat jako parametry rámců a docílit tak zabalení hromadných příkazů a výsledků do jednoho rámce.

5.2.4 Servisní informace

Třída `cService` zajišťuje serializaci struktury `ServerConfig` do zpráv Command a ResultSet v komunikaci TCP. V UDP komunikaci zajišťuje přenos informací mezi servery. Service lze považovat za nástroj pro přenos zpráv distribuujících pohled mezi servery a klienty.
U UDP komunikace je použita samostatně pro jednoduchou a rychlou de/serializaci. Třída zajišťuje rozšíření a aktualizaci pohledu klienta a serveru na ostatní servery. Podle hodnoty čísla `TypeNumber` v hlavičce cService lze určit obsažené servisní informace a obsah deserializovat. Hodnoty `TypeNumber` jsou definovány pro UDP a TCP komunikaci následovně:

- **S_TCP_UPDATE_VIEW, S_UDP_UPDATE_VIEW**
  - **ServerConfig** struktura obsahující všechny informace o serveru.
- **S_TCP_HELLOUS**
  - **T** - hodnota aktuální zátěže serveru.
  - **State** - stav, v jakém se právě server nachází.
  - **IP** - IP adresa určující server.
- **S_UDP_REQUEST, S_UDP_CLOSE**
  - **IP** - IP adresa určující server, který byl ukončen, nebo je žádán zprávou o zaslání pohledu na broadcast.

Pro složitější operace mezi servery (synchronizace apod.) přes TCP předpokládáme přepsání třídy `cService` na samostatný rámec, nebo případný vznik další třídy pro servisní účely mezi servery dědící z `cFrame`.

6 Návrh a implementace serveru

V této kapitole popisujeme návrh a postup implementace aplikace serveru DDS s klientem. Distribuci pohledu na DS mezi uzly a klientem. Dodržujeme námi stanovená pravidla konceptu, viz kapitola 4.3.

6.1 Počátky implementace

Při vývoji složité aplikace jakou je SDDS došlo na počátku implementace hned k několika chybám. Většina chyb pramení z důvodu nepředstavitelnosti celkové kooperativy mezi servery a klienty, statických definic a malé flexibility metod a objektů k použitým DS.

- **Chyby implementace:**
  - Klíč byl tvořen statickým polem hodnot `UInt` s pevně definovanou délkou.
  - (De)Serializace obsahu zpráv probíhala za pomoci staticky definovaných seznamů datových typů a objektů, a to pro jednotlivé typy zpráv.
  - Třída obstarávající přenos zpráv `mySocket` byla navržena bez kontroly přenesených dat a využívala pomalejší blokované spojení TCP.
  - Struktura pro udržování pohledu na servery využívala staticky definovaný klíč, viz výpis 3.
  - Při práci se záznamy a pohledem nebyly použity vždy ukazatele na objekty a struktury.

```
1  typedef struct ServerSettings{
2     char Name[25];
3     unsigned int Id;
4     in_addr Ip;
5     in_addr BroadCast;
6     unsigned short NumRec;
7     float t;//server load or traffic
8     Dimension D[20];
9  }SERVER_SETTINGS;
10
11 typedef struct Dimension{
12   unsigned int DFrom[KEY_DIMENSION];
13   unsigned int DTo[KEY_DIMENSION];
14 }DIMENSION;
```

Výpis 3: Staticky navržená struktura pro pohled na server

- **Úspěchy implementace:**
  - Pro zachycení chování aplikace byl implementován nástroj pro záznam událostí do souboru.

- Implementace a testy mechanizmu pro synchronizaci přístupu k objektům a strukturám.
- Tvorba a řízení vláken pomocí sdílené datové struktury, možnost více vláknového přístupu k serveru.

6.2 DDS v2.0

Aplikace si z předchozích neúspěchů implementace ponechala pouze statické řízení vláken, viz kapitola 6.2.2. Některé nově vzniklé třídy a metody se inspirovaly z původního kódu, avšak vývoj se řídil k větší dynamičnosti a vizí komunikace, viz obrázek 24.

6.2.1 Návrh

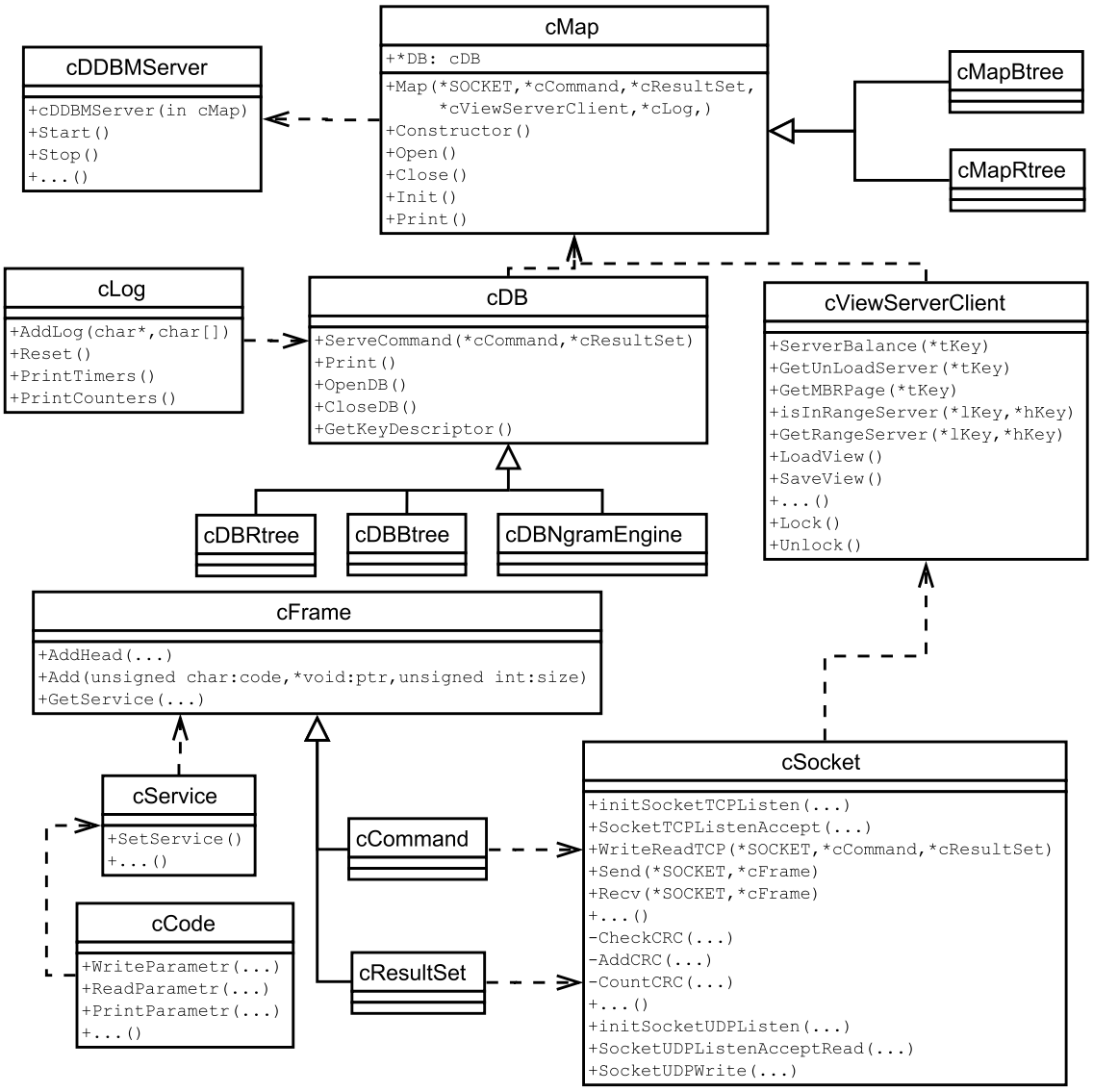
Vznikl dynamický návrh oddělující server s vlákny od zpracování a databáze, viz obrázek 27. S popisem návrhu začneme od třídy `cCode`. Zajišťuje definici (de)serializace tříd a datových typů v rámcích komunikace. Abstraktní třída `cFrame` tvoří základ pro třídy určené k definici komunikačních rámců `cCommand` a `cResultSet`.
Síťovou komunikaci zajišťuje třída `cSocket` implementující metody pro zaslání a získání třídy `cFrame`. Díky dědičnosti lze vytvořit vlastní třídu definující nový komunikační rámec bez zásahu do okolních tříd.
Abychom byli schopni serveru podstrčit jakoukoliv požadovanou DS, vznikla další abstraktní třída `cDB`. Zpracování a udržování pohledu na okolní servery s jejich klíči zajišťuje třída `cViewServerClient`. Definuje metody rozhodující o vyvážení zátěže, možném přeposlání příkazů a adresních chybách klienta.
Různé chování serverů, chcete-li „roli“, při zpracování příkazu zajistí třída dědící z `cMap`. Potomek této třídy definuje, jak má server zpracovat jednotlivé zprávy v metodě `Map()`. Konstruktor serveru tuto třídu vyžaduje jako vstupní parametr.

Poznámka 6.1 Možnost definice chování serverů umožnilo v rámci BP jinému studentovy implementovat a otestovat koncept MapReduce, viz kapitola 3.1.5.

6.2.2 Vlákna

Vlákna jsou v naší aplikaci nepostradatelná. Bohužel mají velký vliv i na náročnost implementace a nutnost zavedení synchronizace přístupu. Aplikace má podle konceptu několik vláken pro jednotlivé funkce serveru.
Zpracování požadavku na server může být realizováno:

a) **Dynamicky** - nové spojení znamená vytvoření nového vlákna pro jeho zpracování. Řešení vyžaduje zavedení sdílené struktury mezi vlákny s informacemi pro jejich řízení. Řízením máme na mysli předání parametrů potřebných pro komunikaci s klientem, zaznamenání spuštěných vláken a vláken připravených k ukončení. Dynamická vlákna si opakovaně vyžádala při testu vkládání na server 0,242 s, oproti tomuto si statická vlákna vystačila s 0,154 s. V obou případech byl použit objekt Mutex [7] pro



Obrázek 27: UML diagram tříd DDS v2.0

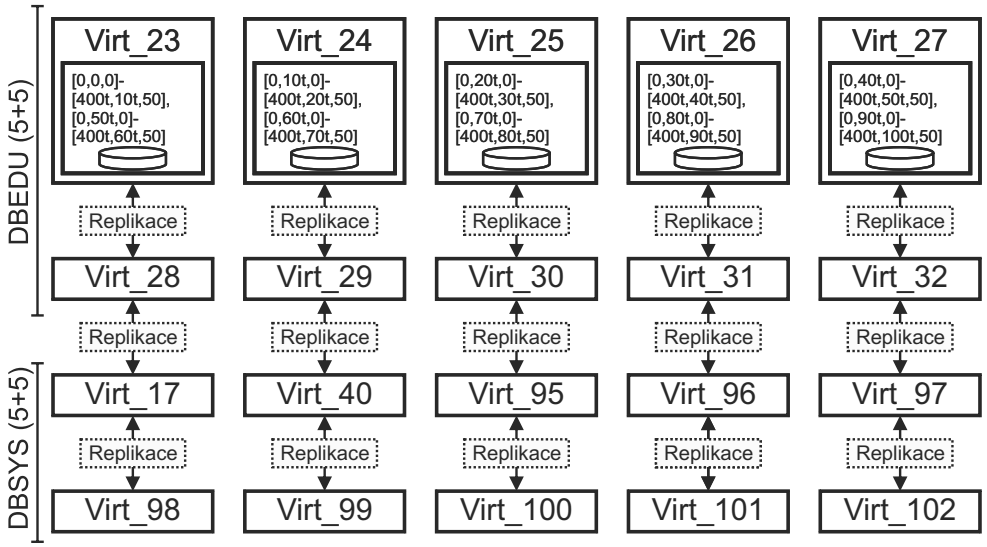
synchronizaci přístupu. Dynamickým vláknům byly potřebné prostředky předem zajištěny - alokovány.

b) **Staticky** - statická vlákna lze naimplementovat snadněji a nemusíme přistupovat ke sdílené struktuře řízení, abychom zaznamenali ukončení vlákna a inicializovali uvolnění prostředků. Vlákna však neustále zatěžují procesor a alokují si neustále i potřebné zdroje. Nemalou výhodou u hromadného zpracování je okamžité zahájení zpracování ve vlákně bez čekání.

Výhody a nevýhody obou přístupů:

- Dynamicky tvořená vlákna
  - Pokud není zapotřebí, nezatěžují procesor ani jiné zdroje.
  - Vhodné pro občasné funkce serveru požadující jedno vlákno.
  - Řízení více vláken vyžaduje více přístupů ke sdílené struktuře.
  - Tvorba a ukončení vlákna s alokací objektů a zdrojů vyžaduje čas.
- Staticky tvořená vlákna
  - Zpracování kódu ve vláknu není zpožděno vytvořením vlákna a potřebných prostředků.
  - Jednoduché a přehledné vytvoření i ukončení vláken.
  - Vhodné při mohutném a častém využití funkce serveru požadující více vláken i současně s důrazem na čas zpracování.
  - Vlákna od okamžiku vytvoření zatěžují procesor.

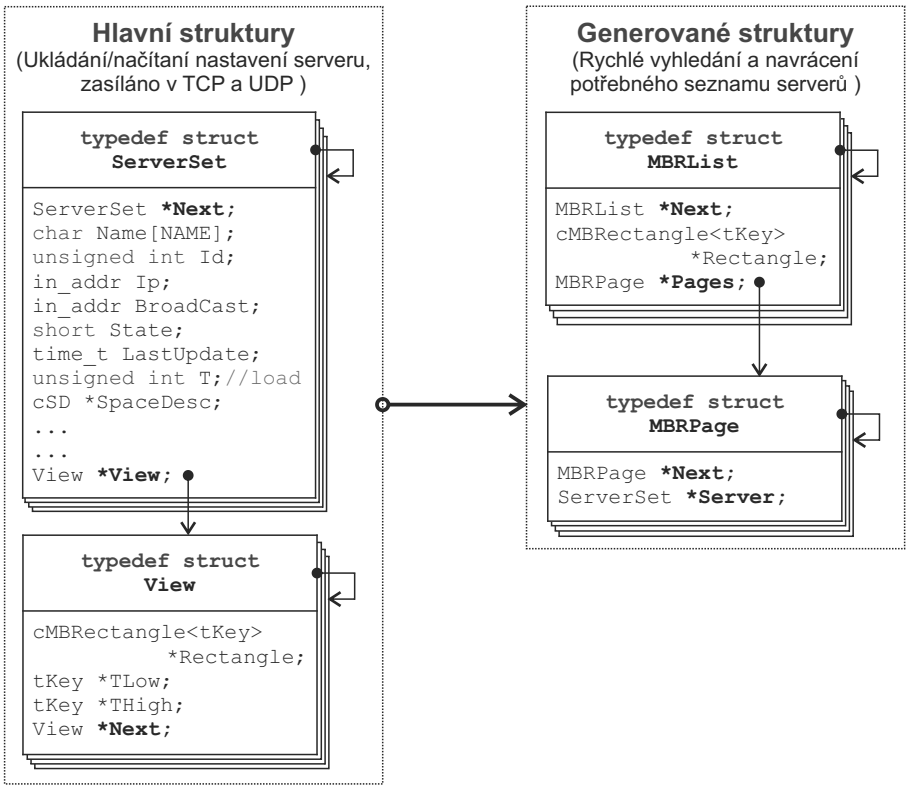
Poznámka 6.2 Vytvoření vlákna zahrnuje alokaci prostředků pro vlákno, proto je vhodné pro dynamickou tvorbu vláken předem alokovat potřebné prostředky a následně je vzniklým vláknům poskytnout.



Obrázek 28: Rozložení databáze DOCWORD na 5-20 virt. serverů

6.2.3 Struktura pohledu

Byl převzat dynamický návrh klíče z implementace DS, viz kapitola 4.2. Využíváme serializovatelnou třídu `cSpaceDescriptor` pro popis datového typu a délky klíče. Následně je použita třída reprezentující v DS objekt *MBR*, viz definice 2.1, udržující dva klíče vzniklé na základě popisu objektem `cSpaceDescriptor`. Tím je zajištěna možnost použití různých klíčů, při zachování tříd s nimi pracujících beze změn.
Pohled na servery tvoří zřetězená struktura `ServerSet` se základními údaji o serveru (IP, jméno, zátěž...) s odkazem na další zřetězený seznam struktur `View` uchovávající informace o rozsahu klíčů na daném serveru.
Často se příkaz potřebuje přeposlat na určitou skupinu serverů. Pro efektivní hledání vznikly dvě další struktury. Tyto struktury vytváří jinak organizovaný přístup ke strukturám `ServerSet` a `View`.



Obrázek 29: Struktury uchovávající nastavení serverů a jejich pohled na okolí

Struktura `cMBRList` slouží jako seznam všech MBR, které jsou mezi servery uchovávány. Má odkaz na zřetězený seznam struktur `cMBRPage` obsahující odkaz na jednotlivé servery. Obě struktury jsou tvořeny pouze ukazateli na již existující informace. Propojení struktur a jejich částečný popis je na obrázku 29. Všechny čtyři struktury jsou společně

s příslušnými metodami k jejich obsluze implementovány ve třídě `cViewServerClient`. Ukázka rozložení rozsahu klíčů na jednotlivé servery je vidět na obrázku 28.

6.2.4 Vyvážení zátěže

Použili jsme metodu vyvažující zátěž serveru zasláním podnětu klientovi o využití jiného serveru se stejnými záznamy [9]. Metodu lze samozřejmě použít jen v případě replikace dat dalším serverem.
Server *S* se zabývá vyvažováním až ve chvíli překročení hranice zátěže. Hranice je definována konstantou v kódu - $CONST\_MIN\_TRAFFIC$ a zabraňuje započetí vyvažování zátěže při velmi nízkých hodnotách. Hranice byla nastavena na hodnotu 300. Podle testů maximálních spojení se serverem za 1s, viz kapitola 6.3.
Při překročení začne server kontrolovat ostatní servery S_{1+n} držící stejný klíč (stejný MBR). V případě nižší zátěže T_{S1+n} vypočítáme aktuální zatížení obou serverů:

$$If\ (T_{S1+n} > CONST\_MIN\_TRAFFIC)\ then\ \frac{T_S + T_{S1+n}}{T_{S1+n}} > BALANCE$$

Prahová hodnota $BALANCE$ = 2,3 zajišťuje započetí vyvažování ve chvíli, kdy server T_{S1+n} má o 30% nižší zátěž než S. Například: T_S = 600; T_{S1+n} = {1 200; 720}

$$If\ (600 > 300)\ then\ \frac{600 + 1\,200}{600} = 3 > 2,3\ (Ano)$$

$$If\ (600 > 300)\ then\ \frac{600 + 720}{600} = 2,2 > 2,3\ (Ne)$$

```
1  ServerConfig * cViewServerClient::GetUnLoadServer(char *Key){
2    if(S->T > CONST_MIN_TRAFFIC){ //min size when use Balance
3      SCBest1 = S;
4      mbrP2 = GetMBRPage(Key);
5      while(mbrP2 != NULL){ //found server with minimal load
6        if(SCBest1 != mbrP2->S && mbrP2->S->State != SERVER_DOWN)
7          if(SCBest1->T > mbrP2->S->T){
8            dif = (SCBest1->T + mbrP2->S->T) / mbrP2->S->T;
9            if(dif > BALANCE) //BALANCE = 2,5
10             if(dif != rand()%(short)dif) //Balancing
11               SCBest1 = mbrP2->S; //used
12         }
13       mbrP2 = mbrP2->Next;
14     }
15     if(SCBest1 != S){
16       return SCBest1;
17     }
18   }
19   return NULL;
20 }
```

Výpis 4: Navržená metoda vyvažující zátěž serveru

Jak je vidět z ukázkového příkladu 3, zašle se klientovi zpět odpověď s informací o serveru T_{S1+n}. Příští dotaz klienta na záznamy držené oběma servery bude směřovat právě na T_{S1+n}. Nevýhodou tohoto způsobu vyvažování je časový odstup, s jakým je informace o zátěži okolních serverů zasílána. Může tak nastat situace, kdy server $S1 + n$ bude v mezičase aktualizace zátěže přes UDP zprávy zatížen více než S.

6.3 Síťová komunikace

Představíme-li si komunikaci dvou stran klient/server, pak je na obou stranách definován objekt Socket. Socket je definován IP adresou, číslem portu a použitým síťovým protokolem. Vytvoření Socketu vyžaduje další parametry určující jednotlivá nastavení spojení na síťové kartě. Na správné implementaci práce se sockety leží potenciál celé síťové komunikace. Implementace DDS počítala s využitím implementace třetí strany. Po testech několika knihoven jsme tuto možnost zavrhli z následujících dvou důvodů.

- Rozsáhlé a dosti nesrozumitelné zdrojové kódy bez bližší dokumentace. Složitá manipulace a případné ladění chyb i výkonu [23].
- Druhým důvodem byla vize možnosti napsat *vlastní třídu/y* právě pro potřeby DDS a zajistit si maximální kontrolu nad nastavením socketu a jeho škálovatelnosti.

Po dlouhé době iteračního vývoje vznikla třída `cSocket`, která obsahuje metody pro práci se sockety protokolu TCP a UDP.
Počátky implementace vznikaly na konceptu komunikace s blokovaným spojením *BLOCKING*. Koncept se z důvodů vyčkávání (časových prodlev) a neustálých problémů při přenosu dat ukázal jako neefektivní. Při volání `connect(), send(), recv()` atpod. dochází k zamrznutí běhu a nemožnosti jej přerušit, dokud není spojení v příslušném stavu. Při volání `recv()` pro větší množství dat ($>$ 64kB) docházelo k předčasnému ukončení přenosu. Došlo k chybě na jedné ze stran a data se nepřenesla korektně. Finální verze třídy řeší všechny tyto problémy. Využívá neblokované spojení - `NON-BLOCKING` s funkcí `select()` pro zjištění stavu navázaného spojení. Provádí se kontrola přenesených dat pomocí CRC, viz kapitola 6.3.4.

6.3.1 Inicializace Winsock DLL

V implementaci je nutné pro možnost použití socketu inicializovat knihovnu Winsock DLL (`Ws2_32.dll`) pro daný proces, viz výpis 5. Knihovna se vyskytuje v několika verzích. V implementaci využíváme nejnovější verzi 2.2. Samotná inicializace probíhá příkazem `WSAStartup()`. Po ukončení práce se sockety, případně chybné inicializaci je nutné zavolat opačnou metodu `WSACleanup()`. V rámci aplikace stačí volat inicializaci/ukončení pouze jednou. Proto jsou obě metody ukryty v konstruktoru a destruktoru třídy.

```
1 bool cmSocket::initWSADLL(void)
2 {
3   WORD wVersionRequested = MAKEWORD(2,2); // Verze
```

```
4   WSADATA data;// structura s informacemi o knihovně lib
5
6   if (WSAStartup(wVersionRequested, &data) != 0)
7   {
8     cerr << "WSAStartup(Socket) inicialization faild." << endl;
9     return false;
10  }
11
12  // byla nalezena použitelná knihovna Winsock DLL?
13  if (LOBYTE(data.wVersion) != 2 || HIBYTE(data.wVersion) != 2)
14  {
15    cerr << "Could not find a Winsock.dll" << endl;
16    WSACleanup();
17    return false;
18  }
19  return true;
20 }
```

Výpis 5: Ukázka inicializace knihovny Winsock

6.3.2 Implementované sockety

V implementaci vznikly metody pro inicializaci a použití následujících socketů v protokolech:

- **Protokol TCP**
  - pro poslech serveru na IP adrese a portu
  - pro akceptování spojení (klient navázal spojení)
  - pro spojení klienta/serveru se serverem
- **Protokol UDP**
  - pro poslech serveru na IP adrese (Broadcast)
  - pro zasílání UDP packetů na IP adresu (Broadcast)

6.3.3 Nastavení socketů

Každá komunikace přes sockety potřebuje specifické nastavení pro správnou funkčnost. Při řešení problémů a maximalizaci výkonu jsme postupně odhalili následující nastavení socketů.

- **FIONBIO** - přepnutí socketu na neblokovaný režim z počátečního blokovaného. Nutno použít při inicializaci naslouchaní serveru na určité IP adrese a portu.
- **SO_LINGER** - nastavení prodlevy mezi voláním `closesocket()` a ukončením/uvolněním socketů. Nutno nastavit u testovacích klientů, aby nevyčerpali všechny adresy portu během testů, viz kapitola 6.4.3.

- **SO_BROADCAST** - nastavení UDP socketu pro odesílání zpráv broadcast.
- **SO_REUSEADDR** - nastavení pro možné okamžité znovupoužití určité IP adresy a portu. Nutno nastavit při inicializaci naslouchání serveru pro případ výpadku serveru a nutnosti rychlé znovu inicializace poslechu na stejné adrese a portu.
- **TCP_NODELAY** - nastavení pro vypnutí algoritmu Nagle na paměti socketu. Vypnutím nedochází k prodlevě při zasílání dat < 1260 B. Buffer se okamžitě vyprazdňuje odesláním dat a nečeká se na plné naplnění rámce.

Pro pohodlné nastavení jednotlivých parametrů socketu a odchycení případných chyb vznikly statické metody, viz výpis 6.

```
1 bool cmSocket::setSocketTCPNoDelay(SOCKET *socket){
2   short iResult;
3   int set = 1;
4   iResult = setsockopt(*socket, IPPROTO_TCP, TCP_NODELAY,(char *)&set,
        sizeof(int));
5   if (iResult == SOCKET_ERROR) {
6     cerr<<"cmSocket::setSocketTCPNoDelay:Setsockopt for TCP_NODELAY failed.
          "<<endl;
7     return false;
8   }
9   return true;
10 }
```

Výpis 6: Ukázka nastavení parametru TCP_NODELAY socketu

6.3.4 Kontrola CRC

Cyklický redundantní součet je metoda k detekci chyb během přenosu zprávy. CRC kód v hlavičce zprávy je implementován ve třídě `cFrame`, z níž zprávy dědí. Potomci třídy, `cCommand` a `cResultSet` pak obsahují kontrolní hodnotu. Po přijetí celé zprávy je hodnota hash spočítána a zkontrolována s hash hodnotou obsaženou ve zprávě. Tím je zajištěn přenos proti možné chybě v důsledku selhání přenosové techniky. Třída *CRCHash.h* [24] s metodou výpočtu byla dodána se zdrojovými kódy databáze.

6.3.5 Testování propustnosti

Na serveru byly zprávy zkontrolovány pomocí metody `CheckCRCHash()` a následně zaslána odpověď o správnosti doručení. Všech vláken na serveru bylo využito ve chvíli, kdy přenos dat na/ze serveru zaměstnal postupně všechna vlákna dřív, než se mohlo některé uvolnit. Sledovat, kdy k tomuto jevu dojde, nebylo možné. Statistika využití vláken proto chybí, avšak testy jsou provedeny pro konfiguraci serveru s 8, 12, 16 a 32 vlákny.

6.3.6 Průběh přenosu

Data se po síti zasílají rozdělené do jednotlivých framů/rámců a jeden obsahuje maximálně 1 260 bajtů. Velikost rámců na síťové vrstvě je dána technologii přenosu - Ethernet (MTU = 1 500B).
Hodnota MTU je nastavena v systému a lze ji změnit na tzv. *Jumbo rámec*. Velikost Jumbo rámce je maximálně 9 000 bajtů užitečných dat. Změna však přináší mnoho problémů v rámci nastavení u klienta a přenosu. Větší velikost MTU nemusí podporovat některý síťový prvek a dojde k fragmentaci rámce na menší nebo k zahození komunikace.
Kontrola komunikace na transportní vrstvě proběhla pomocí programu WireShark [25]. Po zaslání rámce s daty následuje potvrzení [ACK], rozdělení dat po 1 260 B je rozhodující u velkého počtu spojení se soubory do 14 kB. Je zde již zanedbatelná hranice, zda pošlu 11 x 1 260 B nebo vícekrát. Zrychlení a využití přenosového média při právě jednom framu, lze postřehnout v grafu přenosu/s. Zde na křivce testu C1xS1 je efektivněji přeneseno 1 260 B, než 1,5 kB. Data 1,5 kB si vyžádají místo jednoho framu s daty [PSH] a potvrzením [ACK] hned dvojnásobek.
Lze navrhnout následující vzorec pro výpočet celkových nákladů na spojení a přenesená data:

$$[\text{SYN}]192\text{B} + [\mathbf{M} + \lceil \mathbf{M}/1260 \rceil \text{ *2* } [\text{ACK}]54\text{B}] + [\mathbf{N} + \lceil \mathbf{N}/1260 \rceil \text{ *2* } [\text{ACK}]54\text{B}] + [\text{FIN}]174\text{B} \quad (2)$$

Kde M představuje velikost zaslaných dat a N velikost dat přijatých. Výsledek použití tohoto vzorce a nastínění nákladů na přenos je v grafu 32 a 33. Testy vedly k následujícím doporučením pro jedno a vícevláknový server.

Poznámka 6.3 Velikost rámce na 1 GBit lince je shodná s FastEthernet linkou, u obou je použita technologie Ethernet.

6.3.7 Jedno vláknový server

Server s jedním vláknem pro vyřízení navázaných spojení bude procesor v době nečinnosti zatěžovat nejméně. Pouze jedno vlákno vyčkává a dotazuje se, zda není spojení čekající ve frontě v rámci poslechu na IP adrese a portu.
Nemusíme řešit případnou synchronizaci a zamykání objektů, např.: databáze, aby byla zajištěna integrita dat. V době vývoje neměly použité DS implementován zámek pro výcevláknový přístup. Spojení čekají v řadě za sebou a jsou vyřízeny postupně (sekvenčně). Vlákno při zpracování příkazu, mezi přijetím a odesláním odpovědi nechává síťové médium nevyužité a *klesá propustnost*.

- Doporučení pro dosažení maximálních hodnot 90-100 %
  - Maximální počet spojení za sekundu
    - data do velikosti 2 kB (2 rámce po 1 260 B)
  - Maximální vytížení sítě
    - data od 192 kB

6.3.8 Více vláknový server

Server s n>1 vlákny bude zatěžovat procesor postupným dotazováním jednotlivých vláken, zda nečeká ve frontě neobsloužené spojení. Vytvářet vlákna dynamicky podle zátěže i s jejich prostředky (alokace paměti, objekty apod.) by bylo časově náročné. Musíme řešit případnou synchronizaci a zamykání objektů (databáze), aby byla zajištěna integrita dat. Pro zamykání objektů je použit Mutex, který dovoluje v jednu chvíli práci pouze jednomu vláknu s daným zdrojem.
Vlákna jsou časově omezena pouze ve chvíli, kdy se pokouší přistoupit k společným prostředkům, tím však není spojení s klientem. Musí se pouze podělit o šířku přenosového média sítě (propustnost média roste).

- Doporučení pro dosažení maximálních hodnot 90-100 %
  - Maximální počet spojení za sekundu
    - Server s 8 až 16 vlákny
    - Přenášená data do 2 kB (2 rámce po 1 260 B)
  - Maximální propustnos sítě
    - Server s 8 a více vlákny
    - Přenášená data od 8 kB výše

6.3.9 Test síťového prostředí

Prostředí sítě bylo testováno pomocí programu Tamosoft Troughput Test, záznam výsledků je v tabulce 2. Testy se váží na síť tvořenou dvěma PC propojenými switchem s Fast Ethernetem 10/100 Mbps. Maximální propustnost sítě podle testu dosáhla 98,64 Mbps (12,33 MB/s). Naše nejvyšší přenosová hodnota byla 11,199 MB/s. Rozdíl v hodnotách je dán přibližně 7,9 %, což jsou nejnižší náklady na přenos dat. Naše celková maximální propustnost v testech činila 12,089 MB/s.

TCP	**UDP**
TCP Up: 98,64 Mbps (Ave: 98,93)	UDP Up: 0,00 Mbps (Ave: 0,00), Loss: 0,0%
TCP Down: 92,36 Mbps (Ave: 91,30)	UDP Down: 0,00 Mbps (Ave: 0,00), Loss: 0,0%
Round-trip time(ping): 0,7 ms	

Tabulka 2: Výsledky testu síťového prostředí

6.4 Problémy implementace

Během vývoje aplikace a následných testů se objevilo několik problémů, se kterými se při dalším rozvoji této práce můžeme setkat. Jedná se o problémy spojené s alokací paměti, synchronizací, chováním serverů a testovacím prostředím.

6.4.1 Alokace paměti

Vlastní dynamická alokace paměti za pomocí operátorů `new` a `delete` přináší problémy a rizika s možným neuvolněním paměti, ztrátou odkazu na hodnotu. To vše po určitém čase běhu aplikace vede k nárůstu a možnému zahlcení operační paměti.
Při prvotním testování vkládání 100 000 záznamů v cyklu, aplikace vyvolala systémovou chybu $memmory\ leak$ a byla ukončena. Díky neuvolnění cca 60 000 objektů třídy `myPacket`. Ty postupně alokovaly veškerou dostupnou operační paměť. Mezi operátory je nutné hledat i možné podmínky či výstupy z bloku kódu, které by mohly vést k přeskočení příkazu uvolnění paměti `delete`. Alokace paměti ve chvíli ,kdy je potřebná, a následné okamžité zahození je časově náročné.
Řešením je alokovat paměť pro jednotlivé operace a objekty hned při spuštění aplikace a uvolnit ji až při ukončení. Pro zjištění možné chyby neuvolnění paměti je vhodné sledovat stav paměti aplikace za pomoci např. Správce úloh (*Task Manager*) nebo jiné aplikace pro správu systémových prostředků.

6.4.2 Síťové prostředí

Testování v neznámém síťovém prostředí jako je například školní síť je nesnadné. Dostupnosti aplikace na použitých portech mohou bránit kromě firewalů na serverech také aktivní prvky v síti. Aktivními prvky myslíme směrovače a přepínače, na kterých lze aktivovat a nastavit bezpečnostní politiky a pravidla.
Při prvotním testování byla aplikace nahrána na server $dbedu.vsb.cz$ a povoleny porty TCP/UDP. Simulace klientů na $dbedu.vsb.cz$ zasílala data na druhý server umístěný fyzicky mimo školní síť a připojený pomocí VPN. Server replikoval data od klientů zpět na server $dbedu.vsb.cz$. Po určitém počtu odeslaných dat (30-80 tisíc spojení TCP) došlo k nedostupnosti serveru či pádu VPN tunelu do školní sítě.

Příkaz	**Parametr**	**Význam**
$netstat$	$-an$	Výpis všech aktivních (resp. navázaných) síťových spojení na TCP a UDP portech. IP adresy a čísla portů jsou vyjádřeny číselně.
$ipconfig$	$-all$	Výpis síťového nastavení všech existujících síťových adaptérů a připojení.
$tracert$	$< IPadresa >$	Zjištění cesty mezi aktuální stanicí a cílem. Vhodné pro zjištění cesty packetu a možných aktivních zařízení na cestě.
$ping$	$< IPadresa >$	Zjištění dostupnosti serveru či jiného zařízení na síti. Příkaz ping může být na některých zařízeních blokován z bezpečnostních důvodů.

Tabulka 3: Příkazy pro zjištění stavu síťového prostředí

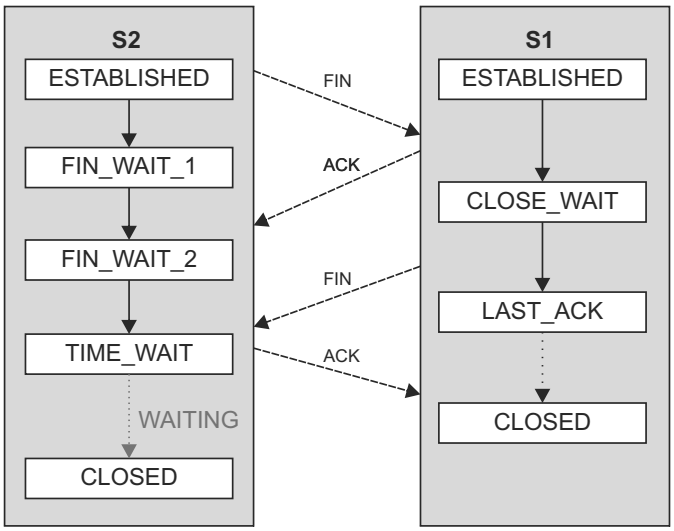
Řešením během testování aplikace na platformě Windows je využít příkazový řádek pro zjištění nezbytných informací o stavu síťové komunikace. Můžeme detekovat zablokování portů, naslouchání serveru na příslušných portech TCP a UDP. Pro zjištění příčin

pádu či občasného selhání komunikace lze využít nástroj WireShark [25]. S jeho pomocí lze zachytit rámce komunikace a podrobit je analýze.

Poznámka 6.4 Testování aplikace by mělo probíhat na předem známém síťovém prostředí.

6.4.3 Dočasná nedostupnost

Při větším přeposílání příkazů (např.: proces replikace záznamů) serverem $S1$ na server $S2$ došlo k dočasné nedostupnosti $S2$ na portu 40 001. Problém zapříčinil koncept ukončení TCP spojení a chování síťové karty.



Obrázek 30: Navázání spojení mezi servery $S1$ a $S2$

Pro možné znovunavázání spojení přejde na určitou dobu stav spojení (session) na serveru $S1$ do stavu *TIME_WAIT*. Standardně je nastavena tato doba až na 2 minuty, po které, i přes příznak *FIN* - ukončení ze strany $S1$, síťová karta $S2$ vyčkává. Vyčkává na možné znovuspojení ze strany $S1$. Při přijetí žádosti o navázání TCP spojení, přiřadí karta unikátní číslo pro dané spojení ze svého rozsahu. Pokud je rozsah vyčerpán mohou nastat tyto dvě situace:

- karta ukončí poslední spojení ve stavu *TIME_WAIT* a založí na tomo portu nové spojení.
- spojení zamítne do doby, než se uvolní některé předchozí spojení.

První spojení klientů s příkazy na síťovou kartu $S2$ vyčerpali rozsah 2^{16} = 65 535 portů a další spojení se již nevytvořila. U karty virtuálního stroje může být maximální počet spojení nižší, v našem případě pouze 5 000 spojení (portů). Jak aktivně ukončit spojení na obou stranách TCP a přimět síťovou kartu k navázání spojení v situaci na obrázku 30?

```
1 struct linger so_linger;
2 so_linger.l_onoff = TRUE;//use defined time
3 so_linger.l_linger = 0;//time in sec
4 iResult = setsockopt(mainSocket, SOL_SOCKET, SO_LINGER,(char *) &so_linger
      , sizeof(so_linger));
5 if (iResult == SOCKET_ERROR) {
6   cout<<"Setsockopt for SO_LINGER failed with error: "<< WSAGetLastError()
        <<endl;
7 }
```

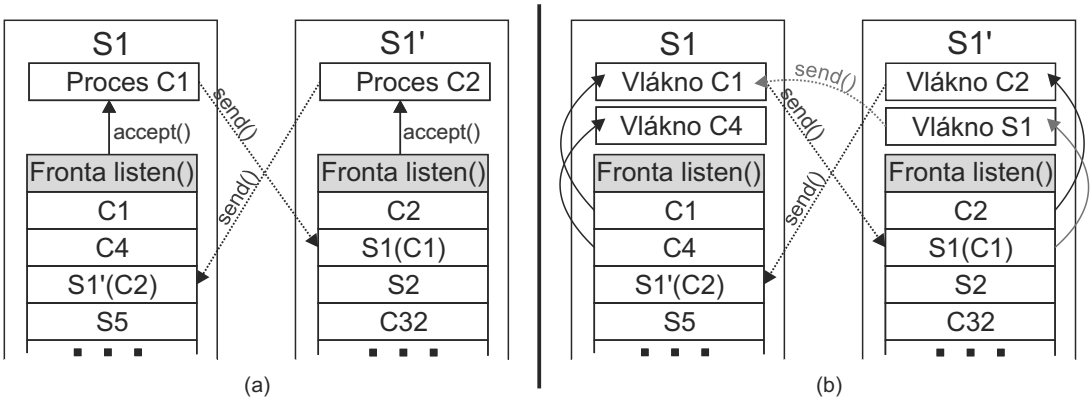
Výpis 7: Úprava parametru SO_LINGER socketu

Řešením je dodatečně socketu při inicializaci nadefinovat tzv. „HARD“ ukončení pomocí metody `setsockopt()` [19]. Spojení nepřejde při volání `closesocket(int socket)` do stavu *TIME_WAIT*, ale bude čekat námi definovanou dobu. Pokud je tato doba $t = 0$, je při volání ukončení spojení ukončeno a přejde ihned do stavu *CLOSED*. Metoda pro nastavení socketu je ve výpise kódu 7.

Poznámka 6.5 Po přidání kódu upravujícího dobu uvolnění spojení se mnohonásobně zrychlila komunikace a uvolnění portu pro další spojení.

6.4.4 Blokování serverů

Testování serverů s nastavením replikace dat může mít za následek postupné zablokování serverů v důsledku čekání na odpověď druhého serveru. Pokud je server pouze jedno vláknová aplikace, vyřizuje jednotlivé požadavky na port 40 001 postupně čtením z fronty pomocí `recv()`. Jak je vidět na obrázku 31 (a), mohou se dva servery navzájem dostat do stavu, kdy oba čekají jeden na druhého. Situace je podmíněna replikací příkazu mezi těmito servery. Nejhorším následkem této situace je ovlivnění celé struktury serverů a jejich lavinové zablokování.



Obrázek 31: Zablokování serverů a možné řešení situace

Poznámka 6.6 Při replikaci dat mezi servery nelze použít server naimplementovaný jako jednovláknová aplikace z důvodu zablokování procesu serverů.

Ostatní servery se nemusí zúčastnit přímo zablokování, ale zastaví se postupně ve frontě těchto serverů. Zablokování se poté šíří s jednotlivými dotazy mezi servery. Existují dvě řešení této situace:

- Omezení doby čekání na odpověď serverů při replikaci dat a opakování transakce. Došlo by tak z některé strany k uvolnění fronty a vyřízení čekajících spojení. Velkou nevýhodou je přílišné zpomalení odezvy serveru a opakování dotazů.
- Použití vícevláknové aplikace serveru. Jednotlivá vlákna zpracují z fronty postupně požadavky i za zablokovaným spojením a následně odblokují vlákno druhého serveru.

Vhodnějším řešením je samozřejmě více vláknová aplikace. Řešení je ilustrováno na obrázku 31 (b), kde server $S1'$ odblokuje zpracováním požadavku $S1(C1)$ z fronty vlákno $S1$.

6.4.5 Synchronizace přístupu

Při implementaci se z důvodu uvedeného v předešlé kapitole 6.4.4, muselo zpracování příkazů přesunout do vláken. Jednotlivá vlákna však nemohou pracovat současně s objekty bez řízení přístupu. Pro synchronizaci byly implementovány zámky nad objekty a strukturami.

```
1 //when init object must create Mutex
2 HANDLE HMutex = CreateMutex(NULL,false,NULL);
3 //destructor~
4 CloseHandle(HMutex);
5 //lock or wait for work with object
6 void myDatabase::Lock(void){
7   DWORD WaitMutex;
8   do{
9     WaitMutex = WaitForSingleObject(this->HMutex,INFINITE);
10    if(WaitMutex == WAIT_OBJECT_0)
11    {
12      break;//take lock, can work with object, data, struct..
13    }else
14      continue;//back to wait
15  }while(true);
16 }
17 //release and unlock object for other waiting thread
18 void myDatabase::UnLock(void){
19  if(!ReleaseMutex(this->HMutex))
20    cerr<<"Can't release mutex in object myDatabase."<<endl;
21 }
```

Výpis 8: Ukázka uzamknutí objektu pro přístup ve vláknech

Zámek je nutné použít i při drobných operacích se sdílenými daty, jako je například pouhá inkrementace proměnné. Před zavedením zámku u sdíleného objektu `cLog` pro statistiky docházelo k nepřesným výsledkům. Důvodem byla inkrementace proměnné v jeden okamžik dvěma a více vlákny - klienty. Zámky k objektům a strukturám vytváříme za pomoci třídy `Mutex`. Mutex může v jednu chvíli „vlastnit" pouze jedno vlákno. Před prací se sdíleným objektem je nutné získat zámek objektu funkcí `Lock()`. Po ukončení práce s objektem je příslušný Mutex uvolněn pomocí `UnLock()`. Kód synchronizace přístupu je na výpise kódu 8.
Nevýhodou zamykaní pomocí Mutexu je zacyklení ostatních vláken volajících `Lock()` na stejný objekt. Vyčkávající vlákna zatěžují procesor neustálým dotazováním o získání zámku pro sebe. Řešení za pomoci uspání těchto vláken příkazem `Sleep(1)` na 1 ms čas uvolnění zámků pouze zhoršil. Pokud vlákno po dokončení své práce nezpřístupní objekt, dojde k nekonečnému čekání všech vláken a zamrznutí aplikace.
Možnou variantou, jak se těmto problémům vyhnout, by bylo zavedení kritických sekcí $CriticalSection$ [22] místo zámků. Otázkou je, jaký vliv by byl na efektivnost aplikace.

Poznámka 6.7 Bez synchronizace přístupu nelze garantovat konzistenci dat.

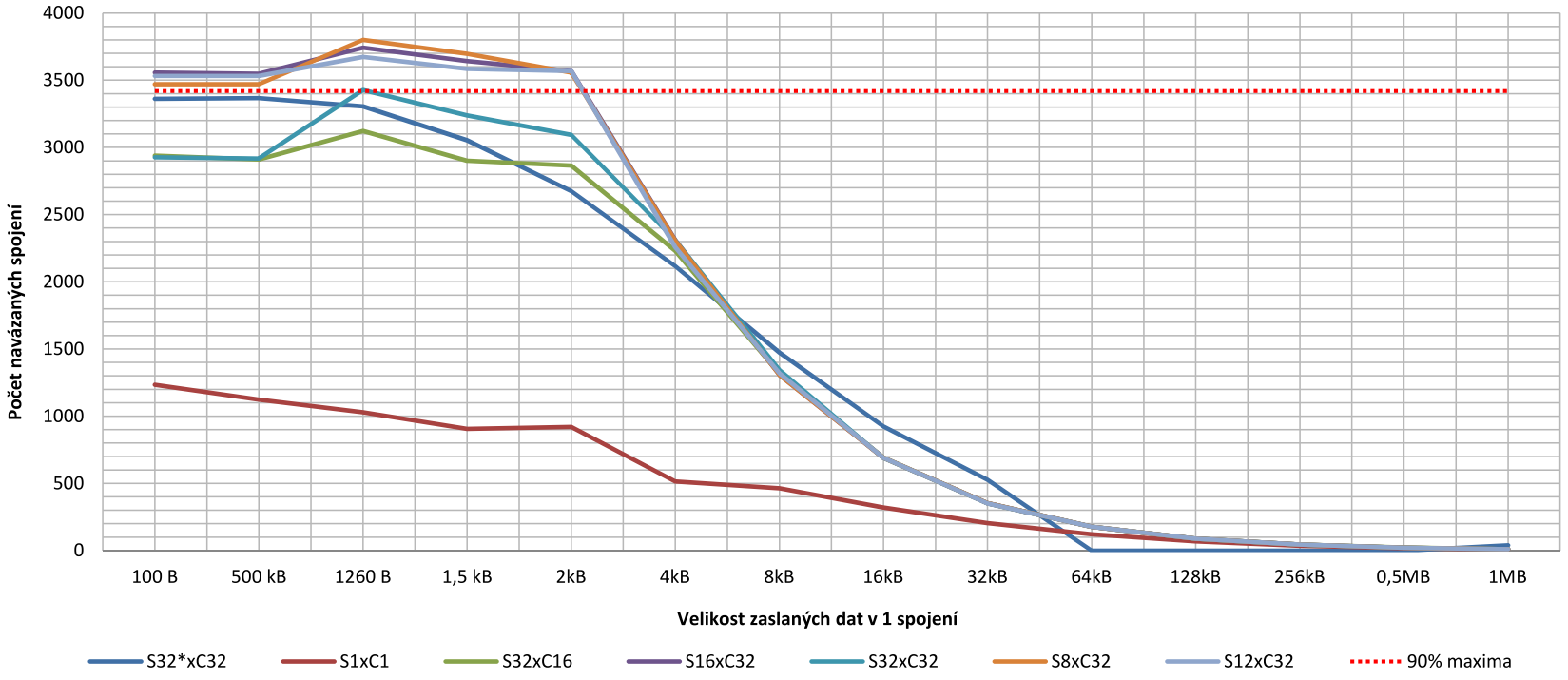
6.4.6 Metered Section

Během testování se vyskytl požadavek na vyzkoušení implementace zámků zvanou $MeteredSection$ [22] k nahrazení Mutexu. Tento nástroj je založen na kritických sekcích a rozšiřuje jeho vlastnosti o počty zdrojů (podobně jako nástroj Semaphore).
Pro test vznikly dvě verze aplikace serveru využívající každý jiný nástroj. Synchronizovaly se struktury pro vlákna, databáze a pohled. Výsledky pro 1 tis. bodových dotazů na server klienty dopadl pro Mutex = 0,355 s a Metered Section = 0,535 s. Zhoršení u Metered Section se pohybovalo podobně i u vkládání a to o 50 %. Z těchto důvodů se od použití Metered Section dále upustilo.

Poznámka 6.8 Výsledky testů synchronizace mohly vzniknout špatnou implementací, případně nevhodným použitím nástroje $MeteredSection$.

Počet spojení za 1s																							
Vláken serevru	Simulovaných klientů (vláken)	Velikost přenesených dat ve zprávě																					
		100 B	500 kB	1260 B	1,5 kB	2kB	4kB	8kB	16kB	32kB	64kB	128kB	256kB	0,5MB	1MB	2MB	3MB	4MB	6MB	8MB	10MB	12MB	14MB
32*	32	3362	3367	3306	3053	2674	2116	1474	924	528	x	x	x	x	40	22	15	11	7	6	4	3	3
1	1	1234	1124	1029	905	920	515	465	321	205	122	68	36	19	9	5	3	2	2	1	1	1	1
32	16	2939	2910	3123	2902	2865	2230	1316	693	353	178	89	45	22	11	5	4	3	2	1	1	1	1
16	32	3558	3548	3741	3642	3567	2309	1306	690	352	178	89	45	22	11	5	4	3	2	1	1	1	1
32	32	2926	2917	3427	3239	3092	2309	1343	689	352	178	89	45	22	11	5	4	3	2	1	1	1	1
8	32	3469	3469	3799	3697	3557	2305	1307	690	353	178	89	45	22	11	5	4	3	2	1	1	1	1
12	32	3532	3532	3672	3585	3567	2258	1318	691	352	178	89	45	22	11	5	4	3	2	1	1	1	1

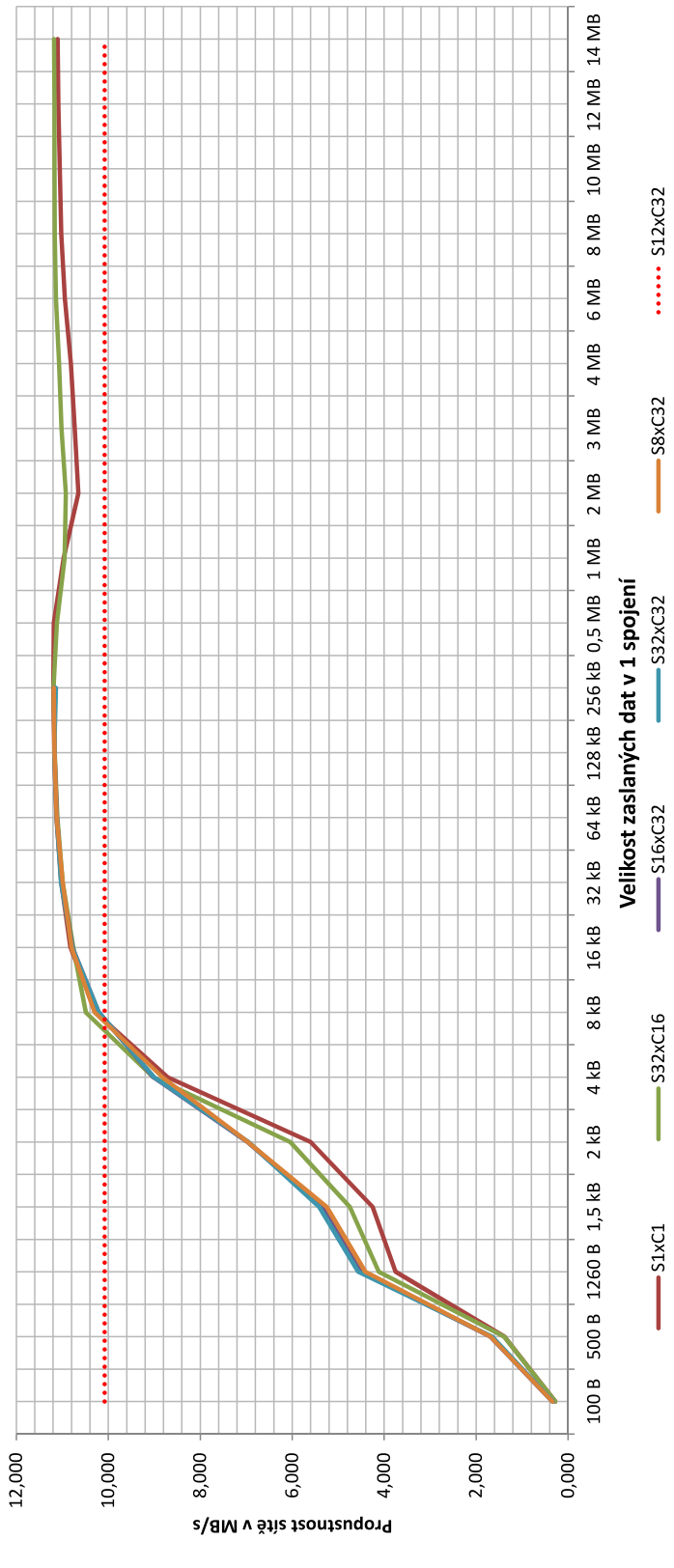
*Test proběhl s umístěním klientů i serveru na jednom počítači - test chování komunikace přes localhost.



Obrázek 32: Tabulka a graf počtu spojení/s

Propustnost sítě v MB/s

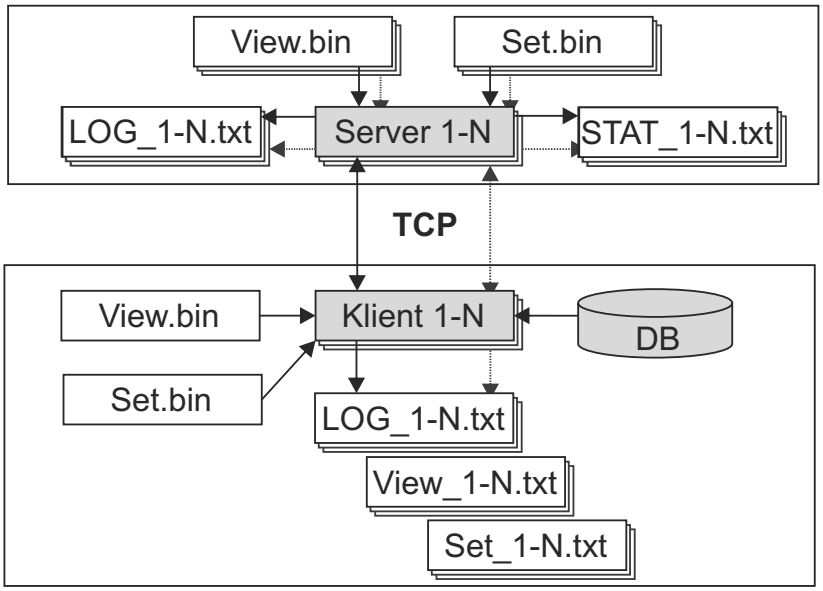
Vláken serevru	Simulovaných klientů (vláken)	Velikost přenesených dat ve zprávě																				
		100 B	500 B	1260 B	1,5 kB	2 kB	4 kB	8 kB	16 kB	32 kB	64 kB	128 kB	256 kB	0,5 MB	1 MB	2 MB	3 MB	4 MB	6 MB	8 MB	10 MB	12 MB	14 MB
1	1	0,120	0,549	1,266	1,326	1,796	2,012	3,629	5,014	6,414	7,612	8,515	9,113	9,331	9,455	9,447	9,515	9,522	9,546	9,582	9,578	9,581	9,583
32	16	0,280	1,388	3,752	4,250	5,595	8,712	10,281	10,823	11,022	11,128	11,175	11,199	11,187	10,972	10,652	10,728	10,815	10,943	11,025	11,055	11,085	11,104
16	32	0,279	1,391	4,118	4,744	6,040	9,021	10,489	10,761	10,995	11,108	11,168	11,194	11,118	10,949	10,923	11,018	11,069	11,139	11,163	11,169	11,169	11,181
32	32	0,339	1,692	4,496	5,335	6,968	9,021	10,205	10,779	11,014	11,119	11,173	11,181										
8	32	0,331	1,654	4,565	5,415	6,947	9,004	10,208	10,781	11,030	11,124	11,175	11,139										
12	32	0,337	1,684	4,413	5,251	6,967	8,821	10,298	10,796	10,991	11,123	11,171	11,187										



Obrázek 33: Tabulka a graf propustnosti přenosu/s

7 Testování

V této kapitole popisujeme jednotlivé testy provedené s první aplikací DDS v1.0 a finální verzí 2.0 na fyzických a virtuálních serverech, dále jen uzly. Nastavení průběhu testu se postupně mění s rostoucím prostředím (počtem uzlů). Pro možnost srovnání jednotlivých výsledků testů se nastavení aplikace neměnilo.
Testy se pokouší poukázat na výhody a nevýhody vyplývající z distribuce a paralelizace záznamů. Rozložení hardwarových prostředků na jednotlivé uzly hraje roli v rámci cenové dostupnosti. Nechceme prověřit pouze funkčnost aplikace, ale nastínit počet uzlů potřebný k dosažení vlastností aplikace srovnatelné s *embedded* řešením. Nabízí se testy s požadavkem zajištění dostupnosti přístupu k datům v případě selhání některého z uzlů.



Obrázek 34: Základní schéma testovaní

7.1 Prostředí testů

Pro testy byly použity servery *DBEDU*, *DBSYS* a *EPE*. Na prvních dvou vznikly postupně virtuální stroje, dále jen uzly, dostupné přes vzdálenou plochu. Testování probíhalo vzdáleně s připojením do školní sítě pomocí VPN tunelu. Na každý virtuální server připadla dvě vlákna procesoru. Fyzické servery *DBEDU*, *DBSYS* a spojuje 100 Mbit switch. Síťové karty na virtuálních serverech jsou nastaveny na 10 Gbit.
V testech je použita datová kolekce DOCWORD s 18 mil. záznamy. Každý záznam je tvořen 3-dimenzionálním klíčem. Rozložení kolekce mezi skupinu 5-ti aplikací DDS je popsáno v tabulce 4.

Statistika DS B-strom					
Výška stromu	3	**Vnitřní kapacita**			127
Velikost položky v listu [B]	17	**Velikost indexu [B]**			16
Kapacita listu	119				
	Virt_23	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Celkem položek	3 164 263	4 308 321	4 031 521	3 969 741	2 830 899
Celkem uzlů	53 501	72 846	68 166	67 120	47 863
Statistika DS R-strom					
Výška stromu	3	**Vnitřní kapacita**			72
Velikost položky v listu [B]	17	**Velikost indexu [B]**			28
Kapacita listu	119				
	Virt_23	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Celkem položek	3 285 884	4 483 584	4 180 658	4 127 739	2 940 694
Celkem uzlů	96 964	134 021	121 375	122 884	86 839

Tabulka 4: DDS v2.0 - Statistika kolekce DOCWORD mezi skupinou uzlů

7.2 Distribuovaný SŘBD

V testech se zpočátku objevuje pro srovnání i první implementace aplikace označená DDS v1.0. Následně se testy zaměřují pouze na výslednou aplikaci DDS v2.0. Použít 32 vláken pro obsluhu spojení na jednom uzlu bylo zvoleno po testech s 15-ti uzly. Kdy při testech docházelo k zablokování jednoho z uzlů a následně kaskádovému zablokování všech ostatních. Doporučení získané z TCP testů třídy `cSocket` a uvedené ve zprávě TCP_Report, použít pro maximální propustnost vícevláknového serveru pouze 8 vláken, nebylo možné.

Poznámka 7.1 Není známo, jak virtuální uzly zprostředkovávají předávání dat mezi virt. síťovými kartami a jaká jsou omezení fyzické síťové karty.

7.3 Klienti

Vytížit jednotlivé uzly nelze aplikací simulující jednoho klienta. Vznikla třída `cTester-Clients` simulující chování více klientů s jedním či více zdroji testovacích dat. Třída vytvoří až 32 vláken představujících jednotlivé klienty. Nastavení a základní pohled si klienti načítají z předem připravených souborů. Vytváří si postupně svůj vlastní pohled na DDS a ukládají si průběh testu do log souborů. Jedna aplikace (32 klientů) se ukázala na dostatečné zatížení nedostatečná, při testech se přikročilo k simulaci až 256 klientů z více aplikací a počítačů.

Název	Operační systém	Procesor	RAM	IP adresa
EPE	WinServer 2003 SP2	XEON E5430 2.66 GHz	8 GB	158.196.128.112
DBEDU	WinServer 2008 R2 SP1	2 x XEON X5670 2.93 GHz (6 jader + Hyper-threading)	96 GB	158.196.141.66
Virt_23	WinServer 2003 SP2	2 vlákna	2 GB	158.196.98.23
Virt_24				158.196.98.24
Virt_25				158.196.98.25
Virt_26				158.196.98.26
Virt_27				158.196.98.27
Virt_28				158.196.98.28
Virt_29				158.196.98.29
Virt_30				158.196.98.30
Virt_31				158.196.98.31
Virt_32				158.196.98.32
DBSYS	WinServer 2008 R2 SP1	2 x XEON E5-2690 2.90 GHz (8 jader + Hyper-threading)	288 GB	158.196.141.12
Virt_17	WinServer 2008 R2 SP1	2 vlákna	2 GB	158.196.98.17
Virt_40				158.196.98.40
Virt_95				158.196.98.95
Virt_96				158.196.98.96
Virt_97				158.196.98.97
Virt_98				158.196.98.98
Virt_99				158.196.98.99
Virt_100				158.196.98.100
Virt_101				158.196.98.101
Virt_102				158.196.98.102

Tabulka 5: Popis dostupného hardwarového vybavení a rozmístění virtuálních uzlů

7.3.1 Umístění klientů

Při testu bodových dotazů nad DBEDU a jeho virt. uzly se ukázal problém s umístěním klientů. Jak je zaznamenáno v tabulce 38, neprojevily se předpokládané vlastnosti DDS, a to zvýšení propustnosti. Naopak se propustnost snížila, špatné výsledky testů lze přisoudit zatížení serveru DBEDU a DBSYS 20-ti virt. uzly a zároveň aplikacím simulující klienty. Změnou umístění klientů na server EPE jsme dosáhli předpokládaného zvětšení propustnosti. V testech se klienti simulovali na fyzických umístěních podle tabulky 6. Bohužel nebylo vždy možné oddělit klienty od DDS, aby nedocházelo k ovlivnění výkonu vinou sdílených hardwarových prostředků.

Poznámka 7.2 Rozdíl v umístění klientů je zřetelný při porovnání tabulky 37 a 38.

Klientů	Umístění	Uzlů	Umístění
4 x 32	DBSYS (4)	5 - 10	DBEDU (5-10)
4 x 32	EPE (4)	15 - 20	DBEDU (10) + DBSYS (5-10)
8 x 32	DBSYS (8)	5 - 10	DBEDU (5-10)
8 x 32	DBSYS (4) + EPE (4)	15 - 20	DBEDU (10) - DBSYS (5-10)

Tabulka 6: Umístění klientů při testech

7.4 Pravidla a záznam testů

Při testech se zapisovaly výsledky z log souborů a výpisů aplikací. Znázornění průběhu testu a vzniklých souborů je na obrázku 34. Pro větší přesnost jsme prováděli 3 měření pro jednotlivé sestavy uzlů a dotazy. V tabulkách testů je uveden průměr testů a podrobný záznam hodnot testu na jednotlivých uzlech. Po testu se aplikace na uzlech vždy ukončila a zapsaly se výsledky z log souborů. Před spuštěním testu je přítomen pouze soubor se základním nastavením serveru. Obsahuje IP adresu a další nastavení uzlu s udržovanou DS. Testovací klienti mají na počátku testu vždy pohled pouze na jeden z uzlů DDS. Ze záznamu testů lze tento uzel(y) identifikovat podle pole „Adresní chyby“.
Klienti se na první uzel obrátí v případě, kdy neznají odpovídající adresu záznamu a vědomě se dopustí adresní chyby. Následně je jím zaslán s odpovědí pohled na nový uzel. Dobře je vidět tento proces například na záznamu testu v tabulce 22

Konstanta	Hodnota	Popis
CONST_MIN_TRAFFIC	200	Minimální hodnota zátěže pro aplikaci metody vyvažování na serveru.
BALANCE	2,3	Práh pro srovnání poměru zátěže mezi serverem a jeho možnou náhradou pro klienta.

Tabulka 7: Nastavení hodnot pro vyvážení zátěže serverů

7.5 Testy embedded DS

Pro provedení testů obou DS v embedded byla implementována testovací aplikace pro zjištění maximální propustnosti při vkládání, bodovém a rozsahovém dotazu. Testy proběhly na serveru DBEDU. Záznamy pro vkládání a dotazy byly připraveny v operační paměti pro dosažení maximální propustnosti.

7.5.1 Výsledek testu

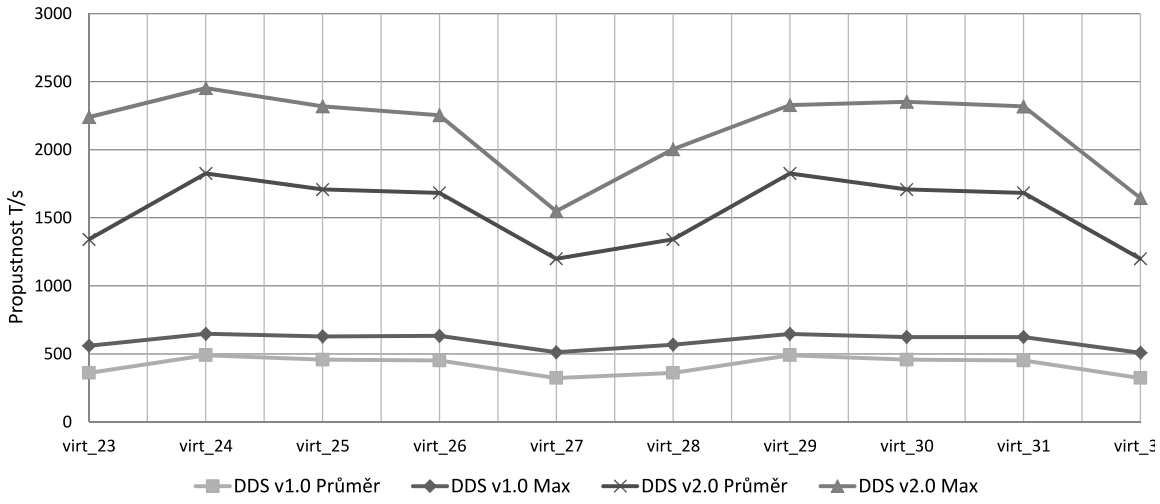
Abnormálně vysoký čas rozsahových dotazů R-stromu je dán nadměrným rozsahem dotazovaného MBR obdelníků. Při testech s menším rozsahem nebyla distribuce na ostatní uzly dostatečná (méně než 5%). Dosahovala časů kolem 60s (59,89s; 58,621s; 59,989s). To odpovídá přibližně 4 až 5-ti násobnému zhoršení oproti B-stromu. Průměrné výsledky jsou uvedeny v tabulce 8 a celý záznam testů je v příloze A v tabulce 16.

DS	Vkládání - 18 mil.		Bodové dotazy - 1 mil.		Rozsahové dotazy - 1 mil.	
	Čas [s]	Propustnost [op/s]	Čas [s]	Propustnost [op/s]	Čas [s]	Propustnost [op/s]
B-strom	5,649	702 780	1,291	776 259	13,139	76 113
R-strom	934,42	19 263	7,919	126 312	1 857,284	539

Tabulka 8: Výsledky propustnosti embedded DS

7.6 První testy implementace

V příloze jsou obsaženy dva záznamy testů. První jsme testovali vkládání a následně bodové dotazy na 10-ti uzlech první implementace DDS v1.0. Pro porovnání propustností se provedly testy se stejnými parametry u DDS v2.0. Na záznamu prvního testu, viz



Obrázek 35: Srovnání propustnosti vkládání DDS v1.0 a v2.0

tabulka 19, lze vyčíst 85 chyb v průběhu testu. Z 18 mil. vkládaných záznamů se vlivem chyb spojení nevložilo či nereplikovalo 85 záznamů. Obrázek 35 nám ukazuje průměrnou a maximální propustnost vkládání na jednotlivých uzlech. Srovnání propustnosti ukazuje nárůst z průměrné hodnoty 416 na 1 551 příkazů za 1s na jednom uzlu při zátěži 32 klientů. Dosahujeme navýšení propustnosti o 273%.

Aplikace	Čas [s]	Dotazů	Klientů	Propustnost [op/s]	Propustnost (1 uzel) [op/s]
DDS v1.0	289,294	1 mil.	32	3 457	343
DDS v2.0	87,797	1 mil.	32	11 390	1 139

Tabulka 9: DDS v1.0 vs. DDS v2.0 - Vkládání na 10 uzlů, B-strom

V tabulce 9 je porovnání výsledku bodových dotazů s navýšením propustnosti o 230%. Z nárůstu propustnosti vyplývá, že jsme se při implementaci nové komunikace aplikace

nedopustili chyb vedoucích k snížení propustnosti oproti první implementaci. Podrobný záznam testů bodových dotazů je v tabulce 19 a 12.

7.7 Testy DDS

V podkapitolách postupně prezentujeme průběh a výsledky testů naší výsledné aplikace při vkládání, bodových a rozsahových dotazech. Poslední provedený test zjišťuje chování klienta při selhání jednoho ze dvou uzlů při komunikaci. Testy probíhaly pro DS B-stromu a R-stromu v sestavě 5 - 20 uzlů, viz obrázek 28. Při pěti uzlech nejsou záznamy replikovány, nedochází k balanci zátěže a distribuci dotazů na ostatní uzly.
Některé testy byly výrazně ovlivněny virtualizací a nedostatkem hardwarových prostředků. Pokud v testech začala propustnost klesat místo očekávaného růstu, nebyly testy s větší zátěží a počtem uzlů provedeny. Výsledky a průměry všech testů jsou uvedeny v tabulce 17.

7.7.1 Vkládání

Testy vkládání lze rozdělit do dvou částí, a to podle cíle testů:

- Zjištění maximální propustnosti vkládání.
- Zjištění propustnosti vkládání při rostoucích nárocích na replikaci záznamů mezi uzly.

První test zaznamenaný v tabulce 22 simuluje vkládání 256-ti klientů ze serveru DBSYS na pět virtuálních uzlů na DBEDU. Při několika testech došlo k nedostupnosti jednoho z virt. serverů. Vkládání záznamu bylo klientem následně opakováno. V žádném z testů nedošlo k situaci nevložení záznamu klientem a nekonzistenci záznamů v DS. Testy zachycující pokles propustnosti jsou v příloze B. Pro srovnání výsledků testů byl simulován stejný počet klientů z DBSYS nebo EPE. Výsledky jsou znázorněny na obrázku 37 a srovnánís embedded v tabulce 10. Klienti znali vždy na začátku testu jen adresu aplikace na uzlu Virt_23, na který provedli vkládání s adresní chybou. Podle počtu klientů lze z tabulek vyčíst počet adresních chyb s přeposláním záznamu a počet aktualizací pohledu klientů. Maximální propustnost vzrostla u B-stromu z 11 717 na 13 920 vložení/s při zvýšení zá-

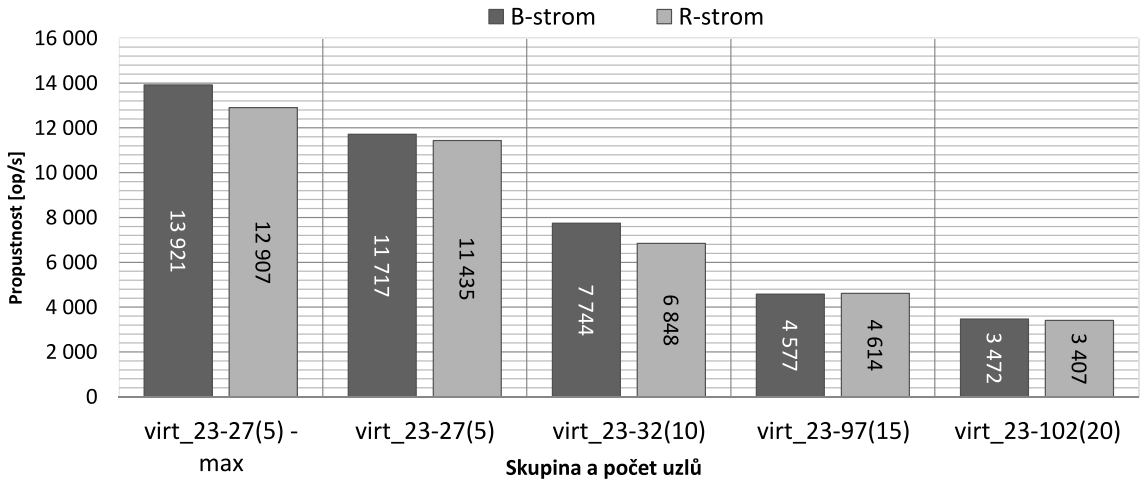
DS	Maximální propustnost [op/s]	Propustnost embedded [op/s]	DDS vs. Embedded	Dosažení embedded řešení	
				1 uzel [op/s]	Celkem uzlů
B-strom	13 920	702 780	50x	2 784	253
R-strom	12 907	19 263	1/3	2 581	11

Tabulka 10: Porovnání propustnosti vkládání

těže z 64 na 128 klientů. Maximální propustnost vkládání 13 921/s u B-stromu je oproti embedded řešení s výsledkem 702 780/s 50-krát pomalejší. Zajímavějšího poměru dosahuje vkládání u R-stromu. Propustnost 12 907/s je oproti embedded s 19 263/s pouze o $^1/_3$

pomalejší. Srovnatelných propustností bez replikace dat s použitím průměrné propustnosti 2 784/s na jeden uzel dosáhneme u B-stromu s 253-mi uzly. R-strom by potřeboval jen 11 uzlů DDS. Rozdíl mezi výsledky, kdy je R-strom pomalejší než B-strom, je dán výkonem DS R-stromu, který podporuje vícerozměrný rozsahový dotaz.

Poznámka 7.3 U vkládání byla zaznamenána nejvyšší propustnost jednoho uzlu ze všech provedených testů, a to 6 334 vložení/s.



Obrázek 36: Graf srovnání propustnosti v závislosti na replikaci

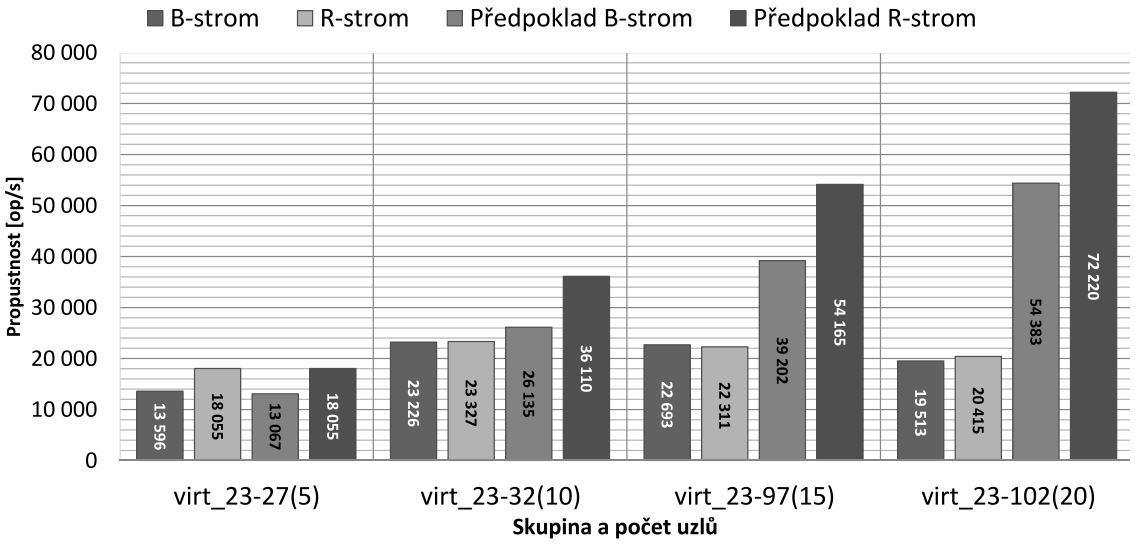
Z grafu je zřejmý pokles propustnosti při replikaci vkládaného záznamu (příkazu `Command`) na ostatní uzel(y). Propustnost neklesá lineárně podílem počtu replikace záznamu. Při první replikaci je pokles průměrně 33% a následně propustnost klesá logaritmicky s počtem replikací.

7.7.2 Bodové dotazy

U testů bylo hlavním záměrem zjistit postupným zvyšováním počtu klientů:

- maximální propustnost uzlů
- schopnost balance zátěže dotazů mezi uzly

Testy proběhly se zatížením uzlů od 128 - 256 klienty. Provedení bodového dotazu nevyžaduje spolupráci ostatních uzlů, vyjma adresní chyby. Z tohoto důvodu uvádíme v grafu i předpoklad zátěže neovlivněné nedostatkem hardwarových zdrojů. Předpokládaná propustnost vznikla z násobků průměrné propustnosti naměřené u Virt_23-27 (5). Maximální dosažené propustnosti v testech jsou zobrazeny v grafu 37. V grafu je nastíněn i předpoklad růstu výkonu plynoucí z maximální naměřené hodnoty propustnosti při 5-ti



Obrázek 37: Graf srovnání maximálních propustnosti bodových dotazů

uzlech. V testech je uveden počet zaslaných pohledů klientům jednotlivými uzly. U sestavy 10-ti až 20-ti uzlů dochází pomocí zasílání pohledu k balanci zátěže a přesměrování klienta na méně vytížený uzel. Bohužel neznáme průběžné zatížení jednotlivých uzlů v průběhu testů. Můžeme pouze porovnat počty dotazů a pohledů odeslaných uzly.
Pokud se podíváme podrobněji na test 20-ti uzlů v tabulce 34, pak 128 klientů obdrželo při 4 mil. dotazů s odpovědí i 2 111 017 pohledů na jiný uzel s nižší zátěží. Zaslání pohledu ve více jak polovině spojení naznačuje neustálé přetížení minimálně jednoho ze 4 uzlů s replikovanými záznamy. Připomeňme, že hranice balance zátěže byla stanovena na rozdíl 30% zátěže. Rozdíl minimálního a maximálního zatížení uzlů se replikací činil od 28,36% k 80,79%. I přes značný počet zaslaných zpráv s vyvážením zátěže není očekávané rozložení zátěže znát ve výsledcích testů. Nejvíce pohledů zaslaly uzly s největším počtem provedených dotazů, a to až v 95,67% odpovědích klientům. Je zřejmé, že se klienti nezachovali podle očekávání, a uzel i přes obdržený pohled zatěžovali dotazy nadále. Testy zachycující všechny bodové dotazy jsou v příloze C.

DS	Maximální propustnost [op/s]	Propustnost embedded [op/s]	DDS vs. Embedded	Dosažení embedded řešení	
				1 uzel [op/s]	Celkem uzlů
B-strom	13 595	776 259	33x	2 719	285 (57 x 5)
R-strom	18 055	126 312	5,5x	3 611	35 (7 x 5)

Tabulka 11: Porovnání propustnosti bodových dotazů

Bodové dotazy jsou oproti embedded řešení u B-stromu 33-krát a u R-stromu 5,5-krát pomalejší. Pokud použijeme propustnost 5-ti uzlů, dostaneme oproti embedded řešení

u B-stromu 57-krát a u R-stromu 7-krát pomalejší propustnost. Srovnatelných výsledků B-stromu s embedded bychom teoreticky dosáhli při 285-ti uzlech. R-strom by potřeboval pouze 35 uzlů. Je otázkou, jaký vliv má na rozložení zátěže 1s zpoždění (interval UDP zpráv) *"Hellou"* při propustnosti uzlu až 5 468 dotazů/s, viz tabulka 36.

DDS v2.0 - Bodové dotazy na 10 uzlů, B-strom					
Základní pohled	Virt_23 + Virt_28				
	Čas [s]	**Dotázaných záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
Průměr	87,797	1 mil.	32	11 390	0
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	56 808	117 377	102 207	131 521	94 877
Dotazů	56 744	117 377	102 207	131 521	94 877
Čas DB [s]	1,138	2,527	2,052	2,542	1,888
Max prop. [op/s]	1 347	1 705	1 840	2 537	1 348
Průměr prop. [op/s]	647	1 337	1 164	1 498	1 081
Pohledů	44 420	10 502	19 465	120 241	15 858
Adresní chyby	64	0	0	0	0
Uzel	**Virt_28**	**Virt_29**	**Virt_30**	**Virt_31**	**Virt_32**
Spojení celkem	116 608	119 034	118 620	85 465	57 651
Dotazů	116 544	119 034	118 620	85 465	57 651
Čas DB [s]	2,454	2,249	2,369	1,957	1,369
Max prop. [op/s]	2 373	1 815	2 252	1 673	875
Průměr prop. [op/s]	1 328	1 356	1 351	970	657
Pohledů	98 373	82 918	97 492	28 752	28 501
Adresní chyby	64	0	0	0	0

Tabulka 12: DDS v2.0 - Bodové dotazy na 10 uzlů, B-strom, 32 klientů

7.7.3 Rozsahové dotazy

Rozsahové dotazy proběhly vždy s použitím jednoho z klíčů DS jako dolní meze. Minimálně byl vždy dotazem vrácen alespoň jeden záznam. Pro horní mez dotazu byl použit stejný klíč s náhodně přičtenou hodnotou v rozsahu 5 000 až 85 000. Rozsah byl zvolen

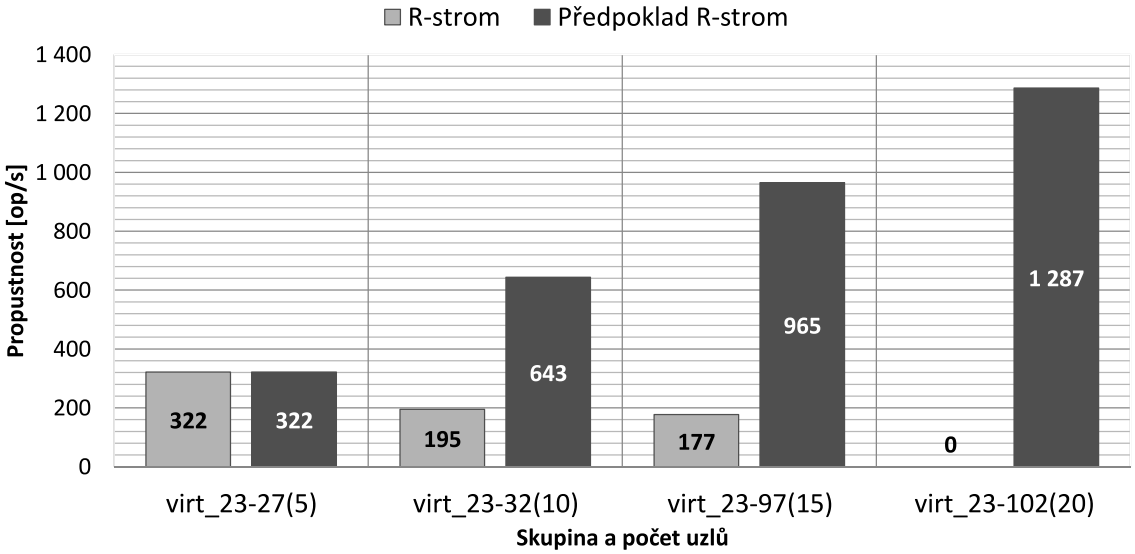
DS	**Maximální propustnost [op/s]**	**Propustnost embedded [op/s]**	**DDS vs. Embedded**	**Dosažení embedded řešení**	
				1 uzel [op/s]	**Celkem uzlů**
B-strom	3 733	76 113	20,6x	747	105 (21 x 5)
R-strom	322	539	1,7x	64	10 (2 x 5)

Tabulka 13: Porovnání propustnosti rozsahových dotazů

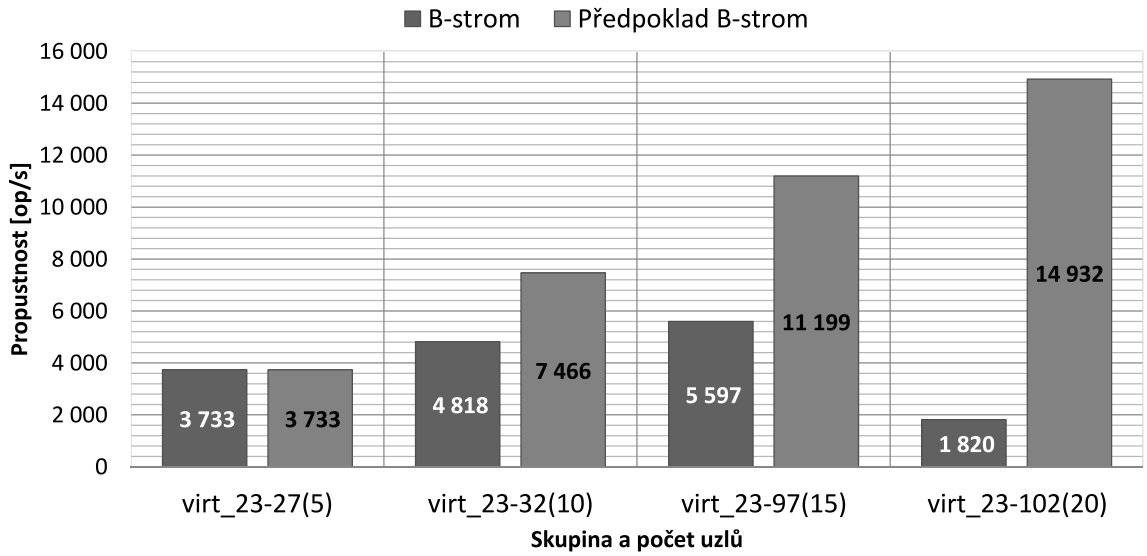
tak, aby došlo i k distribuci dotazu na všechny ostatní uzly.
Výsledky B-stromu vlivem nedostatku zdrojů degradují při testech na 15-ti a 20-ti uzlech. U R-stromu se vlivem náročných rozsahových dotazů projevila degradace již u 10-ti uzlů. Při testech 1 dotaz vyvolal na okolních uzlech v průměru 3 další dotazy. Průměrně uzel získal a vložil do odpovědi 100 záznamů, maximálně bylo dosaženo i 1 000 záznamů v 1 odpovědi klientovi.

DDS v2.0 - Rozsahové dotazy na 5 uzlů, B-strom					
Základní pohled	Virt_23				
Dotazů	1 mil.	**Klientů**	1 x 32	**Chyb**	0
	Čas [s]	**Propustnost [op/s]**	**Výsledků celkem**	**Max výsledků v 1 odpovědi**	**Výsledků**
Průměr	267,890	3 733	204 874 723	1 225	764 827
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	802 298	833 791	821 619	844 550	820 925
Dotazů	802 170	833 791	821 619	844 550	820 925
Replikací dotazu	547 176	682 768	740 621	672 714	479 660
Čas DB [s]	21,815	32,992	31,972	32,022	20,356
Max prop. [op/s]	3 393	3 515	3 492	3 578	3 474
Průměr prop. [op/s]	2 995	3 112	3 067	3 153	3 064
Pohledů	0	32	32	32	32
Adresní chyby	128	0	0	0	0

Tabulka 14: DDS v2.0 - Rozsahové dotazy na 5 uzlů, B-strom a R-strom, 32



Obrázek 38: Graf srovnání rozsahových dotazů R-stromu

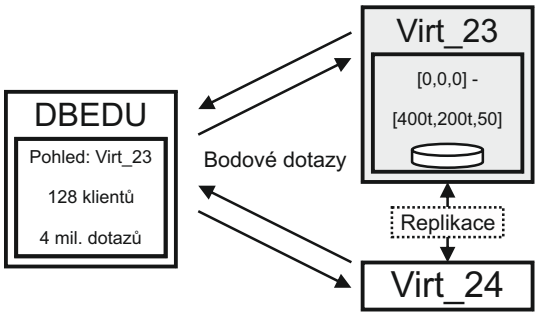


Obrázek 39: Graf srovnání rozsahových dotazů B-stromu

Přibližně $^1/_3$ všech dotazů skončila navrácením pouze jednoho záznamu. Propustnost B-stromu roste s počtem uzlů až k 5 597 dotazům/s. Pro dosažení propustnosti embedded řešení bychom potřebovali 105 uzlů (rozložení 21 x 5). Propustnost R-stromu je maximálně 322 dotazů. Pro dosažení propustnosti embedded řešení bychom potřebovali 10 uzlů. Při použití předpokládané propustnosti 1 287 na 20-ti uzlech překročíme propustnost embedded řešení dvojnásobně. Bohužel se nám vinou nedostatečných hardwarových zdrojů tento výsledek nepovedl naměřit. Testy zachycující rozsahové dotazy jsou v příloze D.

7.7.4 Test dostupnosti DDS

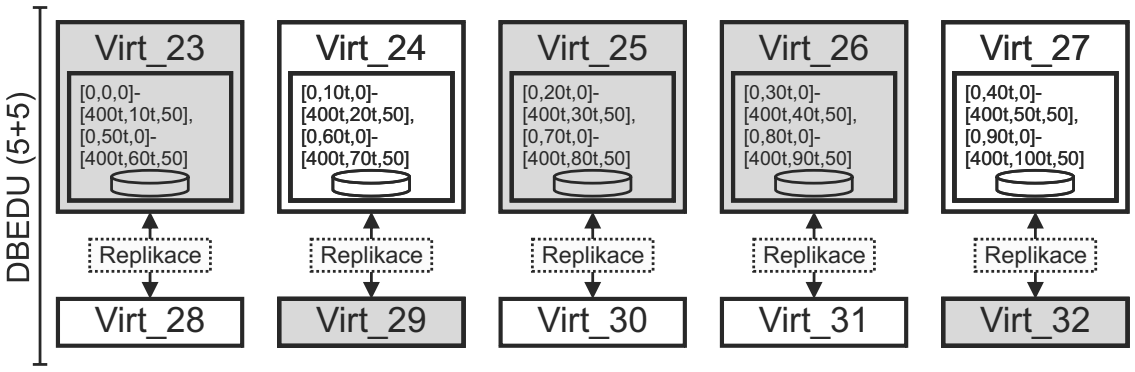
Zrcadlení záznamů na uzlech neslouží pouze pro zvýšení dostupnosti při paralelních dotazech, ale i k zajištění dostupnosti při možném selhání jednoho či více uzlů. Pokoušíme se zjistit průběh a reakci klientů na případné selhání uzlu.



Obrázek 40: Test dostupnosti

První test zjišťuje zajištění dostupnosti při plné replikaci záznamů mezi dva uzly Virt_23 a Virt_24. Na tyto uzly proběhla z DBEDU zátěž tvořená 128-mi klienty s cílem dotázat se na 4 mil záznamů. Základní pohled pro všechny klienty je pouze Virt_23. Po čase se klienti naučí celý pohled, tedy existenci uzlu Virt_24. Po 30s od počátku testu je aplikace na Virt_23 násilně ukončena spolu s probíhajícími spojeními klientů. Pro porovnání jsme otestovali dostupnost i bez selhání.
Doba mezi voláním metody `connect()` na klientovi a vyvoláním chyby ERROR_TCP_ESTABLISH dosahovala při testech až *21s*. Musíme si uvědomit, že se jednalo o nedostupnost cílové IP adresy. Při našem testu zůstane síťová karta v síti dostupná. Pouze aplikace nebude naslouchat na příslušném portu. Klienti se při chybě probíhajícího spojení na uzel Virt_23 pokusí o znovunavázání spojení. Společně s ostatními klienty postupně zjistí nedostupnost uzlu. V pohledech klientů se stav uzlu změní z SERVER_OK na SERVER_DOWN. Veškeré další dotazy jsou adresovány pouze na uzel Virt_24.
Testy proběhly dle očekávání. Klienti s právě probíhajícím spojením na ukončený uzel zahlásili chybu spojení a opakovali svůj dotaz. Při zjištění nedostupnosti uzlu klienty se všechny dotazy začaly směřovat k uzlu Virt_24.

Poznámka 7.4 Nedostupnost uzlu netestujeme při vkládání záznamů klienty, a to z důvodu plné replikace mezi uzly.



Obrázek 41: Test nedostupnosti 5-ti uzlů

Druhý test zjišťuje zajištění dostupnosti při selhání 5-ti uzlů, záznam průběhu testu je v tabulce 15. Kolekce DOCWORD je plně replikována mezi dvě skupiny uzlů Virt_23-27 a Virt_28-32, viz obrázek 41. Na tyto uzly proběhla z DBEDU zátěž tvořená 128-mi klienty s cílem dotázat se na 4 mil záznamů.
Oproti předchozímu testu byl základní pohled klientů na všech 10 uzlů. Nebylo jinak možné zajistit, aby všichni klienti v době ukončení některého z uzlu znali náhradní uzel. Po čase se klienti sice naučí celý pohled, avšak není jasné, kdy k tomu dojde. Po 30s od počátku testu jsme postupně násilně ukončili uzly Virt_23, Virt_29, Virt_25, Virt_26, Virt_32. Ostatních pět uzlů tak stále zpřístupňuje všech 18 mil. záznamů.
Při testu došlo k 337 chybám spojení při komunikaci klienta s ukončeným uzlem. Následně se klienti při novém dotazu nebo jeho opakování pokusili 512-krát navázat kontakt

s již nefungujícím uzlem. Při zjištění nedostupnosti uzlu změnili jeho stav na nedostupný a pokračovali dotazem na zastupující uzel. Na obrázku 41 jsou nedostupné uzly podbarveny šedě. Pokud výsledek testu porovnáme s výsledkem dotazů na uzly bez selhání, dostaneme rozdíl 77,406 s (333,953 - 256,547) s, propustnost dotazů poklesla z 15 592 na 11 978 operací za sekundu.
Test ukázal chybu v procesu adresování uzlů u klienta při nedostupnosti. Pokud mělo 128 klientů při testu pouze základní pohled na uzel Virt_23, pak se několik z nich nenaučilo před selháním celý pohled. Toto zjištění vyplývá ze souborů obsahujících pohled testovacích klientů po testu bez selhání. Klienti se jeden či dva uzly nenaučili. Následně při selhání uzlů došlo u některých k neschopnosti dotázat se na data. Přitom nejspíše znali alespoň jeden z uzlů, který nebyl ukončen. Pokud by na něj provedli dotaz (adresní chybu), poskytl by klientovi v odpovědi informaci o fungujícím uzlu se záznamy.
Místo toho klient našel záznam o uzlu držící daný klíč, prvním dotazem zjistil, že je uzel nedostupný a pozměnil si jeho stav ve svém pohledu. Následně v cyklu opakování dotazu našel ve svém pohledu znovu stejný uzel ve stavu nedostupný. O navázání spojení s uzlem se nepokusil a po 10-ti neuspěšných pokusech oznámil chybu nedostupnosti uzlu. V závěru práce navrhujeme úpravu části klienta, která se zabývá adresací uzlu.

DDS v2.0 - Bodové dotazy na 10 uzlů, B-strom					
Základní pohled	Virt_23 - Virt_28				
	Čas [s]	**Dotázaných záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
Průměr	333,953	4 x 1 mil.	4 x 32	11 978	337
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	X	942 404	X	X	616 979
Dotazů	X	942 404	X	X	616 979
Čas DB [s]	X	22,980	X	X	15,658
Max prop. [op/s]	X	5 718	X	X	3 716
Průměr prop. [op/s]	X	2 821	X	X	1 848
Pohledů	X	358 154	X	X	327 905
Adresní chyby	0 - plný pohled klientů				
Uzel	**Virt_28**	**Virt_29**	**Virt_30**	**Virt_31**	**Virt_32**
Spojení celkem	533 187	X	504 876	493 013	X
Dotazů	533 187	X	504 876	493 013	X
Čas DB [s]	14,292	X	14,400	14,262	X
Max prop. [op/s]	4 217	X	4 698	4 804	X
Průměr prop. [op/s]	1 597	X	1 512	1 476	X
Pohledů	368 306	X	335 977	45 656	X
Adresní chyby	0 - plný pohled klientů				

Tabulka 15: DDS v2.0 - Bodové dotazy na 10 uzlů s nedostupností, Bstrom, 4 x 32

8 Závěr

Začátek práce se věnuje vlastnostem jednotlivých SDDS s popisem způsobů distribuce a rozložení dat. Uvedené struktury umožňují paralelní zpracování, zvýšení propustnosti a duplicitu dat mezi uzly. Těchto vlastností je dnes zapotřebí u aplikací s důrazem na dostupnost a s velkým počtem klientů.
V rámci vývoje navrhujeme ze získaných poznatků vlastní koncept DDS s distribucí a rozložením pohledu na data. Serializaci volání metod a komunikaci jsme z počátku chtěli převzít z veřejně dostupných API knihoven. Následně jsme se rozhodli pro vlastní implementaci. Navrhli a implementovali jsme metodu vzdáleného volání metod za pomoci dvou příkazů *Command* a *ResultSet*. Zde jsme vycházeli z konceptu navrhnutého společně s vedoucím práce. Třídy aplikace implementujeme v jazyce C++. Aplikaci serveru s nově vzniklou třídou pro komunikaci jsme otestovali na reálné síti s použitím TCP a UDP protokolů. Z výsledků jsme vytvořili report s doporučeními pro správné použití a nastavení aplikace serveru.
Implementace aplikace s sebou přinesla i mnoho problémů, které jsme v průběhu práce řešili. Jedním z problémů byla synchronizace přístupu vláken ke sdíleným objektům. Provedli jsme testy synchronizace za pomocí nástroje Mutex a Metered Section. Po srovnání rychlosti synchronizace jsme použili rychlejší nástroj Mutex.
Výsledkem práce je aplikace vícevláknového serveru a klienta s možností využití pro různé datové struktury. Reálné použití našla aplikace v projektu SGS Detekce plagiovaných dokumentů. Zpřístupňuje rozhraní DS pomocí webového klienta.
V závěru práce se zabýváme srovnáním naší aplikace a embedded řešením za použití DS B-stromu a R-stromu. Testy probíhaly na třech školních serverech s dalšími 20-ti virtuálními uzly. Přístup k serverům probíhal s využitím VPN tunelu do školní sítě a vzdálené plochy. Pozdější výsledky testů ukazují degradaci propustnosti s rostoucím zatížením. Degradaci přisuzujeme nedostatečným hardwarovým prostředkům pro jednotlivé uzly, dělení o síťovou kartu a procesor. Nedosáhli jsme předpokládaných násobků propustnosti při replikaci dat.
Při vkládání se projevilo očekávané snížení propustnosti s rostoucí replikací dat mezi uzly. Výsledky bodových dotazů poukázaly na úměrný růst propustnosti s počtem replikací a rozsahové dotazy se dosti přiblížily propustnosti embedded řešení. Abychom dosáhli propustnosti embedded řešení, potřebovali bychom pro B-strom 285 (57 x 5) uzlů a pro R-strom 35 (7 x 5). Úspěšně jsme otestovali zajištění dostupnosti dat při výpadku jednoho ze dvou plně replikovaných uzlů. U testu s replikací mezi dvě skupiny uzlů se nám podařilo při nedostupnosti poloviny uzlů zachovat dostupnost všech záznamů. Konvergence klientů na záložní uzel byla okamžitá. Odhalili jsme chybu v procesu výběru uzlu při dotazování klienta v situaci s nedostupností a navrhli jsme způsob řešení.
S rostoucím vývojem Internetu a nároků na zpracování velkého množství dat by bylo vhodné ve vývoji a testech SDDS pokračovat. Provádění testů upřednostnit na fyzická zařízení a nalézt lepší způsoby vyvážení zátěže. Budoucí implementace by se měla zaměřit na způsoby udržení konzistence dat možnou synchronizací replikovaných záznamů DS po výpadku jednoho z uzlů. Některé vlastnosti nebylo možné v rámci diplomové práce plně implementovat. Uvádíme několik nápadů a možných vylepšení aplikace pro do-

sažení vyšší škálovatelnosti, dynamičnosti a rychlosti. Vzniklé prvky aplikace našly již využití i v jiných pracích a aplikacích. Nedostatky a možná vylepšení aplikace:

- Přepsání struktur tvořících pohled do tříd s využitím `template <class tkey>`. Dosáhli bychom větší dynamičnosti klíče a mohli bychom nahradit objekt `cMBRectangle`. Ten nyní slouží i pro popis intervalu klíčů jednorozměrné DS B-stromu.
- Udržování struktur pohledu ve stromové či jiné vhodné struktuře pro vyhledávání. Nyní je pohled tvořen čtyřmi strukturami, které jsou organizovány jako dynamické pole struktur. Vyhledání záznamu uzlu přes klíč tak probíhá sekvenčně.
- Úprava procesu dotazování klienta při zjištění nedostupnosti uzlu v replikovaném prostředí tak, aby se nedopustil opakování dotazu v cyklu a nezískání záznamu. Místo toho by se měl, podobně jako při úplné neznalosti umístění klíče, dopustit adresní chyby na některý jiný uzel. V odpovědi by získal pohled na uzel, který replikuje záznamy nedostupného uzlu, viz kapitola 7.7.4.
- Implementace log souboru pro synchronizaci replikovaných uzlů během výpadku sítě či serveru a zajištění konzistence dat.
- Třídu cSocket doplnit o podporu protokolu IPv6.
- Další testy zámku Critical section nebo jiných synchronizačních nástrojů (události, semafory atp.) s otestováním rychlosti zpracování požadavku, viz kapitola 6.4.5.
- Přepsání vláken aplikace serveru do jiného konceptu řízení, test dynamické tvorby vláken podle potřeby s cílem snížení zátěže CPU.
- Implementace UDP komunikace uzlů bez opakování zpráv. Okamžitá reakce na přijaté zprávy, viz kapitola 4.5.3.
- Implementace automatického rozložení dat mezi uzly a dosažení škálovatelné DDS. Nedefinovali bychom rozsah/interval dat udržovaných jednotlivými uzly, ale pouze klíč záznamů. Uzly by si následně pomocí UDP zpráv mohly rozdělit a určit udržovaný obsah v DS. Tím bychom docílili vyvážení datové zátěže na uzlech.
- Návrh pro vylepšení stávajícího vyvážení dotazů mezi uzly s replikací dat. Případné testy s přepsáním stavající reakce klienta na obdržení zprávy (IAM) o přetížení.
- Řešení hromadných transakcí zabalením více příkazů do jednoho. V rámci implementace lze vložit do `cCommand` jako parametry další serializované objekty `cCommand`, viz kapitola 5.2.

Za pomoci tříd zajišťujících komunikaci klient/server vznikl prostředek pro testování a nasazení DS. Umožňující vyšší dostupnost a škálovatelnost než umožňují embedded datové struktury. V průběhu vypracování DP vznikla k projektu *SGS Detekce plagiovaných dokumentů* DLL knihovna pro C# obsahující komunikaci a zpracování objektů DS. Byla zde

prokázána jednoduchost použití knihovny při implementaci webového rozhraní klienta. Komunikace přes webovou aplikaci klienta na server a zpracování získaných výsledků splnila základní požadavky a dále se testuje.
Zdrojové kódy pro komunikaci s aplikací serveru dala možnost implementace MapReduce prostředí s SQL databází v rámci jiné DP. Je předpokládán další rozvoj a použití vzniklých tříd v jiných aplikacích.

Aleš Nedbálek

9 Reference

[1] WITOLD Litwin, NEIMAT Marie-Anne, SCHNEIDER A. Donovan, LH*—A Scalable, Distributed Data Structure, *Transactions and Database Systems*, 1996, ACM: 0362-5915/96/1200-0480, s. 480-525

[2] WITOLD Litwin, RISCH Tore, LH*g : A High-Availability Scalable Distributed, *Knowledge and Data Engineering, IEEE Transactions*, 2002, ISSN: 1041-4347

[3] LITWIN Witold , M.A. NEIMANT, High-Availability LH* Schemes width Mirroring, *Cooperative Information Systems, Proceedings., First IFCIS International Conference*, 1996, ISBN: 0-8186-7505-5

[4] LITWIN Witold, NEIMAT Marie-Anne, LEVY G., NDIANYE S., SECK T., LH*s : a High-availability and High-security Scalable Distributed Data Structure, *Research Issues in Data Engineering, Seventh International Workshop*, 1997, ISBN: 0-8186-7849-6

[5] LITWIN Witold, Rim Moussa, SCHWARZ Thomas, S.J, LH*g : A LH*RS – A Highly-Available Scalable Distributed Data Structure, *ACM Transactions on Database Systems (TODS)*, 2005, s. 769-811

[6] SCHWARZ Thomas, TSUI Peter, LITWIN Witold, An Encrypted, Content Searchable Scalable Distributed Data Structure, *Data Engineering Workshops, 22nd International Conference*, 2006, ISBN: 0-7695-2571-7

[7] Official Microsoft site, *Mutex – třída*, http://msdn.microsoft.com/cs-cz/library/system.threading.mutex.aspx [cit. 12.8.2012]

[8] du MOUSA C., LITWIN Witold, RIGAUX Philippe, SD-Rtree: A Scalable Distributed Rtree, *Data Engineering, IEEE 23rd International Conference*, 2007, s. 296-305, ISBN: 1-4244-0803-2

[9] LUKAWSKI Grzegorz, SAPIECHA Krzysztof, Balancing Workloads of Servers Maintaining Scalable Distributed Data Structures, *International Euromicro Conference on Parallel, Distributed and Network-Based Processing*, 2011, s. 80-84, ISBN: 978-1-4244-9682-2

[10] YUNMO Chung, Dynamic Signature Hashing, *Computer Software and Applications Conference, COMPSAC 89,* 1989, s. 257-262, ISBN: 0-8186-1964-3

[11] LEWIS Ted G., COOK Curtis R., Hashing for Dynamic and Static Internal Tables, *Computer,* 1988, s. 45-56, ISSN: 0018-9162

[12] Introduction to SDDS, http://ceria.dauphine.fr/Sddstal1.ppt [cit. 12.4.2012]

[13] PRATA Stephen, *Mistrovství v C++, 2. aktualizované vydání,* Computer Press, a.s., Brno, 2004, 1006s, ISBN: 802-5100987.

[14] KRÁTKÝ Michal, *Databázové a informační systémy*, Ostrava, [pdf dokument], 4.3.2007

[15] WU Sai, JIANG Dawei, OOI Beng Chin, KUN-LUNG Wu, Efficient B-tree Based Indexing for Cloud Data Processing, *Conference In VLDB*, 2010, s. 1207-1218

[16] Wikipedie, *TCP/IP*, http://cs.wikipedia.org/wiki/TCP/IP [cit. 25.1.2012]

[17] ANDRZEJAK, A., GOMES J. B., Parallel MapReduce in Python in Ten Minutes, *Data Mining Workshops (ICDMW), 2012 IEEE 12th International Conference*, 2012, s. 402-407, ISBN: 978-1-4673-5164-5

[18] Baselinemag, *How Google Works*, http://www.baselinemag.com/c/a/Infrastructure/How-Google-Works-1/P [cit. 24.2.2012]

[19] MSDN Microsoft Developer Support, *Winsock Reference*, http://msdn.microsoft-.com/en-us/library/windows/desktop/ms741416%28v=vs.85%29.aspx [cit. 26.3.2012]

[20] JAGADISH H. V., OOI Beng Chin, RINARD Martin, BATON: A Balanced Tree Structure for Peer-to-Peer Networks, *Conference In VLDB*, 2005, s. 661-672

[21] Wikipedie, *BATON Overlay*, http://en.wikipedia.org/wiki/BATON_Overlay [cit. 25.3.2012]

[22] CHOU Dan, *A Quick and Versatile Synchronization Object*, MSDN Microsoft Developer Support, http://msdn.microsoft.com/en-us/library/ms810428.aspx [cit. 2.2.2013]

[23] ZeroC, *ICE - Internet Communications Engine*, http://www.zeroc.com/index.html [cit. 21.10.2012]

[24] Database Research Group, *DBRG Framework API 20120313*, Katedra informatiky, VŠB-TU Ostrava, http://db.cs.vsb.cz/api/ [cit. 2.10.2012]

[25] Riverbed Technology, *WireShark*, http://www.wireshark.org/ [cit. 6.1.2013]

[26] KITSUREGAWA, M., TANAKA H., MOTO-OKA T., Architecture and performance of relational algebra machine GRACE, *Conference on Parallel Processing*, 1984

[27] DBC/1012 data base computer concepts and facilities, *International Conference on Very Large Data Bases*, Teradata Document C02-001-05, 1988

[28] DEWITT, D., GERBER, R., GRAEFE, G., HEYTENS, M., KUMAR, K., AND MURALIKRISHNA, M., GAMMA: A high performance dataflow database machine, *Conference In VLDB*, 1986

[29] WITOLD Litwin, NEIMAT Marie-Anne, SCHNEIDER A. Donovan, RP*: A Family of Order Preserving Scalable Distributed Data Structures, *In Proceedings of the Twentieth International Conference on Very Large Databases*, s. 342–353, 1994

[30] SEVERANCE, C., PRAMANIK, S., AND WOLBERG, P., Distributed linear hashing and parallel projection in main memory databases, *Conference In VLDB*, 1990

A Výsledky embedded a DDS

Vkládání - 18 mil. záznamů kolekce DOCWORD				
	B-strom		R-strom	
Test č.	**Čas [s]**	**Propustnost [op/s]**	**Čas [s]**	**Propustnost [op/s]**
1	27,019	666 198	938,406	19 181
2	24,840	724 638	932,101	19 311
3	25,087	717 503	932,754	19 298
Průměr	25,649	702 780	934,420	19 263
Bodové dotazy - 1 mil.				
Test č.	**Čas [s]**	**Propustnost [op/s]**	**Čas [s]**	**Propustnost [op/s]**
1	1,338	747 384	7,863	127 178
2	1,327	753 580	8,110	123 305
3	1,208	827 815	7,785	128 452
Průměr	1,291	776 259	7,919	126 312
Rozsahové dotazy - 1 mil.				
Test č.	**Čas [s]**	**Propustnost [op/s]**	**Čas [s]**	**Propustnost [op/s]**
1	13,040	76 687	1849,490	541
2	13,270	75 358	1830,472	546
3	13,107	76 295	1891,890	529
Průměr	13,139	76 113	1857,284	539
Průměr vrácených záznamů v 1 dotazu		208,4		7,5

Tabulka 16: Výsledky testů embedded datové struktury

Klienti/ uzly	B-strom			R-strom		
Vkládání - 18 mil. záznamů kolekce DOCWORD						
	Čas [s]	**Propustnost [op/s]**	**Chyb**	**Čas [s]**	**Propustnost [op/s]**	**Chyb**
32/2	4 144,945	4 343	0	-	-	-
Test s replikací záznamů na oba uzly.						
128/5	1 293,109	13 921	0	1 394,610	12 907	0
Test maximální propustnosti 5-ti uzlů.						
32/5	1 537,203	11 710	0	1 574,088	11 435	0
32/10	2 324,479	7 744	0	2 630,610	6 843	0
32/15	3 933,135	4 577	0	3 901,517	4 614	0
32/20	5 184,460	3 472	0	5 285,025	3 407	0
Zjištění závyslosti propustnosti na replikaci.						
Bodové dotazy - 1 mil.						
	Čas [s]	**Propustnost [op/s]**	**Chyb**	**Čas [s]**	**Propustnost [op/s]**	**Chyb**
32/10	87,797	11 390	0	-	-	-
Bodové dotazy - 4 mil.						
128/2	381,691	10 480	0	-	-	-
128/2	524,104	7 632	0	-	-	-
Speciální test s replikací záznamů na obou uzlech se selháním uzlu.						
128/5	321,075	12 458	0	316,991	12 619	0
128/10	256,547	15 592	0	250,385	15 975	0
128/15	227,276	17 603	0	235,461	17 071	0
128/20	213,228	18 769	0	208,346	19 239	0
Bodové dotazy - 8 mil.						
256/5	588,441	13 596	2 890	443,153	18 055	2 200
256/10	344,538	23 226	1 704	343,459	23 327	2 409
256/15	353,077	22 693	0	359,979	22 311	0
256/20	424,814	19 513	0	399,880	20 415	0
Rozsahové dotazy - 1 mil.						
	Čas [s]	**Propustnost [op/s]**	**[záznamů/s]**	**Čas [s]**	**Propustnost [op/s]**	**[záznamů/s]**
32/5	267,890	3 733	764 827	3 108,846	322	2 017
32/10	243,154	4 113	845 461	5 137,033	195	1 301
32/15	290,522	3 442	695 660	5 652,437	177	1 069
Rozsahové dotazy - 2 mil.						
128/10	416,296	4 818	972 492	-	-	-
128/15	358,162	5 597	1 117 001	-	-	-
Rozsahové dotazy - 4 mil.						
128/20	2 198,210	1 820	225 940	-	-	-

Tabulka 17: Výsledky všech testů na DDS v2.0

B Tabulky vkládání

DDS v1.0 - Vkládání na 10 uzlů					
Základní pohled	Virt_23				
B-strom					
Test č.	**Čas [s]**	**Použito záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
1	8 645,57	18 mil.	32	2 082	85
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Vloženo	3 110 436	4 237 537	3 963 520	3 903 833	2 784 675
Replikováno	3 110 445	3 972 644	3 838 338	3 414 868	2 091 385
Spojení celkem	3 110 511	4 237 576	3 963 553	3 903 873	2 784 719
Čas DB [s]	46,959	66,118	56,583	55,576	44,119
Max propustnost [op/s]	560	647	628	632	512
Průměr prop. [op/s]	360	490	458	451	322
Pohledů	64	12	14	14	18
Adresní chyby	64	12	14	14	18
Uzel	**Virt_28**	**Virt_29**	**Virt_30**	**Virt_31**	**Virt_32**
Vloženo	3 110 436	4 237 537	3 963 520	3 903 833	2 784 675
Replikováno	0	264 915	125 199	488 987	693 309
Spojení celkem	3 110 499	4 237 585	3 963 564	3 903 874	2 784 734
Čas DB [s]	45,1010	65,4360	58,2280	59,3130	43,5130
Max propustnost [op/s]	567	646	624	623	508
Průměr prop. [op/s]	360	490	458	451	322
Pohledů	0	0	0	0	6
Adresní chyby	0	0	0	0	6

Tabulka 18: DDS v1.0 - Vkládání na 10 uzlů, B-strom

DDS v1.0 - Bodové dotazy na 10 uzlů					
Základní pohled	Virt_23				
B-strom					
Test č.	**Čas [s]**	**Použito záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
1	289,294	1 mil.	32	3 457	0
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Bodový dotaz	172 609	186 628	191 905	155 886	113 787
Spojení celkem	172 675	186 646	191 923	155 904	113 799
Čas DB [s]	2,2570	2,5700	2,8750	2,3090	1,6940
Max propustnost [op/s]	796	879	913	735	576
Průměr prop. [op/s]	596	645	633	539	393
Pohledů	64	16	16	16	10
Adresní chyby	64	16	16	16	10
Uzel	**Virt_28**	**Virt_29**	**Virt_30**	**Virt_31**	**Virt_32**
Bodový dotaz	**0**	51 245	27 107	61 290	39 544
Spojení celkem	**0**	51 247	27 115	61 292	39 546
Čas DB [s]	**0s**	1,0570	0,5170	1,1810	0,4720
Max propustnost [op/s]	**0**	338	149	329	207
Průměr prop. [op/s]	**0**	177	94	212	137
Pohledů	0	0	0	6	0
Adresní chyby	0	0	0	6	0

Tabulka 19: DDS v1.0 - Bodové dotazy na 10 uzlů, B-strom, 4 x 32

DDS v2.0 - Vkládání na 2 uzly					
Základní pohled	Virt_23				
B-strom					
Test č.	**Čas [s]**	**Použito záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
1	4 144,945	18 mil.	32	4 343	0
Uzel			**Virt_23**		**Virt_24**
Spojení celkem			18 000 000		18 000 000
Záznamů v DB			18 000 000		18 000 000
Čas DB [s]			295,523		255,902
Max propustnost [op/s]			5 873		5 855
Průměr propustnost [op/s]			4 344		4 344
Pohledů			5 438 406		3 791 514
Chyb přeposlání			0 - plná replikace dat		

Tabulka 20: DDS v2.0 - Vkládání na 2 uzly, B-strom

DDS v2.0 - Vkládání na 5 uzlů					
Základní pohled	Virt_23				
B-strom					
	Čas [s]	**Vloženo záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
Průměr	1 537,203	18 mil.	32	11 710	0
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	3 111 712	4 236 595	3 964 403	3 903 652	2 783 771
Záznamů v DB	3 111 583	4 236 595	3 964 403	3 903 652	2 783 771
Čas DB [s]	19,558	19,997	19,964	18,084	17,669
Max prop. [op/s]	3 436	4 500	4 032	4 072	2 936
Průměr prop. [op/s]	2 649	3 607	3 375	3 323	23 70
Pohledů	0	32	32	32	32
Adresní chyby	128	0	0	0	0
R-strom					
	Čas [s]	**Vloženo záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
Průměr	1 574,088	18 mil.	32	11 435	0
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	3 111 712	4 236 595	3 964 403	3 903 652	2 783 771
Záznamů v DB	3 111 583	4 236 595	3 964 403	3 903 652	2 783 771
Čas DB [s]	241,552	398,442	370,523	371,274	261,989
Max prop. [op/s]	2 498	3 240	2 984	2 966	2 200
Průměr prop. [op/s]	1 977	2 691	2 519	2 480	1 768
Pohledů	0	32	32	32	32
Adresní chyby	128	0	0	0	0

Tabulka 21: DDS v2.0 - Vkládání na 5 uzlů, R-strom a B-strom

DDS v2.0 - Vkládání na 5 uzlů					
Základní pohled	Virt_23				
B-strom					
	Čas [s]	**Vloženo záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
Průměr	1 293,109	18 mil.	4 x 32	13 921	0
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	3 112 095	4 236 594	3 964 402	3 903 651	2 783 770
Záznamů v DB	3 111 583	4 236 594	3 964 402	3 903 651	2 783 770
Čas DB [s]	77,582	49,658	72,297	76,052	44,313
Max prop. [op/s]	4 859	**6 334**	5 759	5 950	4 173
Průměr prop. [op/s]	2 407	3 276	3 066	3 019	2 153
Pohledů	0	128	128	128	128
Adresní chyby	512	0	0	0	0
R-strom					
	Čas [s]	**Vloženo záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
Průměr	1 394,61	18 mil.	4 x 32	12 907	0
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	3 112 095	4 236 594	3 964 402	3 903 651	2 783 770
Záznamů v DB	3 111 583	4 236 594	3 964 402	3 903 651	2 783 770
Čas DB [s]	792,093	1 220,63	1 136,95	1 164	639,076
Max prop. [op/s]	3 550	4 658	4 777	4 339	2 992
Průměr prop. [op/s]	2 232	3 038	2 843	2 799	1 996
Pohledů	0	128	128	128	128
Adresní chyby	512	0	0	0	0

Tabulka 22: DDS v2.0 - Vkládání na 5 uzlů, R-strom a B-strom, 4 x 32

DDS v2.0 - Vkládání na 10 uzlů					
Základní pohled	Virt_23				
B-strom					
	Čas [s]	**Vloženo záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
Průměr	2 324,479	18 mil.	32	7 744	0
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	3 111 711	4 236 594	3 964 402	3 903 651	2 783 770
Záznamů v DB	3 111 583	4 236 594	3 964 402	3 903 651	2 783 770
Čas DB [s]	79,126	87,478	72,866	83,781s	60,459
Max prop. [op/s]	2 240	2 452	2 319	2 253	1 548
Průměr prop. [op/s]	1 341	1 826	1 708	1 682	1 199
Pohledů	0	32	32	32	32
Adresní chyby	128	0	0	0	0
Uzel	**Virt_28**	**Virt_29**	**Virt_30**	**Virt_31**	**Virt_32**
Spojení celkem	3 111 583	4 236 595	3 964 402	3 903 651	2 783 770
Záznamů v DB	3 111 583	4 236 594	3 964 402	3 903 651	2 783 770
Čas DB [s]	56,753	77,923	79,175	78,657	55,467
Max prop. [op/s]	2 003	2 328	2 352	2 318	1 644
Průměr prop. [op/s]	1 341	1 826	1 708	1 682	1 199
Pohledů	0	0	0	0	0
Adresní chyby	0	0	0	0	0
R-strom					
	Čas [s]	**Vloženo záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
Průměr	2 630,610	18 mil.	32	6 843	0
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	3 111 711	4 236 594	3 964 402	3 903 651	2 783 770
Záznamů v DB	3 111 583	4 236 594	3 964 402	3 903 651	2 783 770
Čas DB [s]	314,225	499,626	447,937	459,247	266,056
Max prop. [op/s]	1 894	2 242	2 185	2273	1653
Průměr prop. [op/s]	1 183	1 610	1 507	1 484	1 058
Pohledů	1 051 356	1 581 978	165 245	238 646	1 217 543
Adresní chyby	128	0	0	0	0
Uzel	**Virt_28**	**Virt_29**	**Virt_30**	**Virt_31**	**Virt_32**
Spojení celkem	3 111 583	4 236 594	3 964 402	3 903 651	2 783 770
Záznamů v DB	3 111 583	4 236 594	3 964 402	3 903 651	2 783 770
Čas DB [s]	304,202	476,636	454,622	396,797	320,446
Max prop. [op/s]	1 790	2 429	2 237	2 263	1 674
Průměr prop. [op/s]	1 183	1 610	1 507	1 484	1 058
Pohledů	534 733	634 863	1 853 682	1 758 886	214 353
Adresní chyby	0	0	0	0	0

Tabulka 23: DDS v2.0 - Vkládání na 10 uzlů, B-strom a R-strom

DDS v2.0 - Vkládání na 15 uzlů					
Základní pohled	Virt_23				
B-strom					
	Čas [s]	**Vloženo záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
Průměr	3 933,135	18 mil.	32	4 577	0
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	3 111 711	4 236 594	3 964 402	3 903 651	2 783 770
Záznamů v DB	3 111 583	4 236 594	3 964 402	3 903 651	2 783 770
Čas DB [s]	75,498	84,076	77,167	66,824	62,145
Max prop. [op/s]	1 396	1 826	1 722	1 764	1 358
Průměr prop. [op/s]	791	1 077	1 008	993	708
Pohledů	1 535 021	1 881 185	228 766	217 445	1 245 972
Adresní chyby	128	0	0	0	0
Uzel	**Virt_28**	**Virt_29**	**Virt_30**	**Virt_31**	**Virt_32**
Spojení celkem	3 111 583	4 236 594	3 964 402	3 903 651	2 783 770
Záznamů v DB	3 111 583	4 236 594	3 964 402	3 903 651	2 783 770
Čas DB [s]	61,459	81,51	77,869	73,171	66,591
Max prop. [op/s]	1 342	1 894	1 793	1 737	1 341
Průměr prop. [op/s]	791	1 077	1 008	993	708
Pohledů	50 548	0	1 548 486	0	80 090
Adresní chyby	0	0	0	0	0
Uzel	**Virt_17**	**Virt_40**	**Virt_95**	**Virt_96**	**Virt_97**
Spojení celkem	3 111 583	4 236 594	3 964 402	3 903 651	2 783 770
Záznamů v DB	3 111 583	4 236 594	3 964 402	3 903 651	2 783 770
Čas DB [s]	182,645	185,161	116,281	180,852	80,211
Max prop. [op/s]	1 345	1 876	1 681	1 816	1 314
Průměr prop. [op/s]	791	1 077	1 008	993	708
Pohledů	0	422 222	260 849	2 640 200	51 023
Adresní chyby	0	0	0	0	0

Tabulka 24: DDS v2.0 - Vkládání na 15 uzlů, B-strom

DDS v2.0 - Vkládání na 15 uzlů					
Základní pohled	Virt_23				
R-strom					
	Čas [s]	**Vloženo záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
Průměr	3 901,517	18 mil.	32	4 614	0
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	3 111 711	4 236 594	3 964 402	3 903 651	2 783 770
Záznamů v DB	3 111 583	4 236 594	3 964 402	3 903 651	2 783 770
Čas DB [s]	311,768	405,116	343,323	352,899	261,017
Max prop. [op/s]	1 172	1 584	1 388	1 372	1 043
Průměr prop. [op/s]	798	1 086	1 016	1 001	714
Pohledů	1 489 477	2 179 128	0	0	1 372 498
Adresní chyby	128	0	0	0	0
Uzel	**Virt_28**	**Virt_29**	**Virt_30**	**Virt_31**	**Virt_32**
Spojení celkem	3 111 538	4 236 594	3 964 402	3 903 651	2 783 770
Záznamů v DB	3 111 538	4 236 594	3 964 402	3 903 651	2 783 770
Čas DB [s]	297,055	365,445	365,997	391,036	294,95
Max prop. [op/s]	1 160	1 574	1 386	1 390	1 054
Průměr prop. [op/s]	798	1 086	1 016	1 001	714
Pohledů	659	2 702	1 985 735	1 983 392	50 272
Adresní chyby	0	0	0	0	0
Uzel	**Virt_17**	**Virt_40**	**Virt_95**	**Virt_96**	**Virt_97**
Spojení celkem	3 111 538	4 236 594	3 964 402	3 903 651	2 783 770
Záznamů v DB	3 111 538	4 236 594	3 964 402	3 903 651	2 783 770
Čas DB [s]	768,342	591,168	611,705	588,057	395,698
Max prop. [op/s]	1 147	1 564	1 413	1 387	1 003
Průměr prop. [op/s]	1798	1 086	1 016	1 001	714
Pohledů	57 178	21	0	67 801	3 006
Adresní chyby	0	0	0	0	0

Tabulka 25: DDS v2.0 - Vkládání na 15 uzlů, R-strom

DDS v2.0 - Vkládání na 20 uzlů					
Základní pohled	Virt_23				
B-strom					
	Čas [s]	**Vloženo záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
Průměr	5 184,46	18 mil.	32	3 472	0
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	3 111 711	4 236 594	3 964 402	3 903 651	2 783 770
Záznamů v DB	3 111 583	4 236 594	3 964 402	3 903 651	2 783 770
Čas DB [s]	72,365	81,815	75,109	79,894	61,392
Max prop. [op/s]	964	1 248	1 030	1 028	800
Průměr prop. [op/s]	600	817	765	753	537
Pohledů	1 572 504	1 957 371	1 893 293	1 931 515	1 314 535
Adresní chyby	128	0	0	0	0
Uzel	**Virt_28**	**Virt_29**	**Virt_30**	**Virt_31**	**Virt_32**
Spojení celkem	3 111 583	4 236 594	3 964 402	3 903 651	2 783 770
Záznamů v DB	3 111 583	4 236 594	3 964 402	3 903 651	2 783 770
Čas DB [s]	64,646	77,071	71,963	80,214	57,683
Max prop. [op/s]	963	1 248	1 047	1 080	804
Průměr prop. [op/s]	600	817	765	753	537
Pohledů	174	932	6 562	7 580	0
Adresní chyby	128	0	0	0	0
Uzel	**Virt_17**	**Virt_40**	**Virt_95**	**Virt_96**	**Virt_97**
Spojení celkem	3 111 583	4 236 594	3 964 402	3 903 651	2 783 770
Záznamů v DB	3 111 583	4 236 594	3 964 402	3 903 651	2 783 770
Čas DB [s]	245,172	255,278	166,569	244,609	204,215
Max prop. [op/s]	944	1 267	1 041	1 031	870
Průměr prop. [op/s]	600	817	765	753	537
Pohledů	14 824	122 770	110 547	8 717	31
Adresní chyby	128	0	0	0	0
Uzel	**Virt_98**	**Virt_99**	**Virt_100**	**Virt_101**	**Virt_102**
Spojení celkem	3 111 583	4 236 594	3 964 402	3 903 651	2 783 770
Záznamů v DB	3 111 583	4 236 594	3 964 402	3 903 651	2 783 770
Čas DB [s]	188,929	189,757	248,794	176,445	117,326
Max prop. [op/s]	891	1 268	1 044	1 036	785
Průměr prop. [op/s]	600	817	765	753	537
Pohledů	0	0	0	0	0
Adresní chyby	128	0	0	0	0

Tabulka 26: DDS v2.0 - Vkládání na 20 uzlů, B-strom

DDS v2.0 - Vkládání na 20 uzlů					
Základní pohled	Virt_23				
R-strom					
	Čas [s]	**Vloženo záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
Průměr	5 285,025	18 mil.	32	3 407	0
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	3 111 711	4 236 594	3 964 402	3 903 651	2 783 770
Záznamů v DB	3 111 538	4 236 594	3 964 402	3 903 651	2 783 770
Čas DB [s]	321,893	372,2	344,256	353,005	276,316
Max prop. [op/s]	913	1 119	1 016	1 046	754
Průměr prop. [op/s]	589	802	750	739	527
Pohledů	1 492 516	1 844 121	2 077 057	1 886 614	1 374 862
Adresní chyby	128	0	0	0	0
Uzel	**Virt_28**	**Virt_29**	**Virt_30**	**Virt_31**	**Virt_32**
Spojení celkem	3 111 538	4 236 594	3 964 402	3 903 651	2 783 770
Záznamů v DB	3 111 538	4 236 594	3 964 402	3 903 651	2 783 770
Čas DB [s]	289,034	381,184	339,535	396,316	268,111
Max prop. [op/s]	878	1 179	997	1 001	750
Průměr prop. [op/s]	589	802	750	739	527
Pohledů	730	124 523	1 056	0	0
Adresní chyby	0	0	0	0	0
Uzel	**Virt_17**	**Virt_40**	**Virt_95**	**Virt_96**	**Virt_97**
Spojení celkem	3 111 538	4 236 594	3 964 402	3 903 651	2 783 770
Záznamů v DB	3 111 538	4 236 594	3 964 402	3 903 651	2 783 770
Čas DB [s]	1 042,49	1 006,453	953,511	949,691	622,183
Max prop. [op/s]	883	1 143	1 049	1 017	742
Průměr prop. [op/s]	589	802	750	739	527
Pohledů	6	213 196	36	171 605	1 886
Adresní chyby	0	0	0	0	0
Uzel	**Virt_98**	**Virt_99**	**Virt_100**	**Virt_101**	**Virt_102**
Spojení celkem	3 111 538	4 236 594	3 964 402	3 903 651	2 783 770
Záznamů v DB	3 111 538	4 236 594	3 964 402	3 903 651	2 783 770
Čas DB [s]	710,307	1 019,08	844,04	883,02	594,202
Max prop. [op/s]	897	1 142	1 017	1 014	742
Průměr prop. [op/s]	589	802	750	739	527
Pohledů	0	0	0	0	0
Adresní chyby	0	0	0	0	0

Tabulka 27: DDS v2.0 - Vkládání na 20 uzlů, R-strom

C Tabulky bodových dotazů

DDS v2.0 - Bodové dotazy na 2 uzly					
Základní pohled	Virt_23				
B-strom					
	Čas [s]	**Dotázaných záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
Průměr	381,691	4 x 1 mil.	4 x 32	10 480	0
Uzel			**Virt_23**		**Virt_24**
Spojení			1 678 043		2 321 957
Dotazů			1 678 043		2 321 957
Čas DB [s]			29,541		37,612
Max prop. [op/s]			5 736		6 682
Průměr prop. [op/s]			4 396		6 083
Pohledů			1 112 621		0
Chyb přeposlání	0 - plná replikace dat				

Tabulka 28: DDS v2.0 - Bodové dotazy na 2 uzly, B-strom, 4 x 32

DDS v2.0 - Bodové dotazy na 2 uzly s nedostupností					
Základní pohled	Virt_23				
B-strom					
	Čas [s]	**Dotázaných záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
Průměr	524,104	4 x 1 mil.	4 x 32	7 632	61
Test dostupnosti při selhání 1 uzlu					
Uzel			**Virt_23**		**Virt_24**
Spojení			X		3 693 129
Dotazů			306 871		3 693 129
Čas DB [s]			X		61,747
Max prop. [op/s]			X		9 500
Průměr prop. [op/s]			X		7 047
Pohledů			X		2 049 601
Chyb přeposlání	0 - plná replikace dat				

Tabulka 29: DDS v2.0 - Bodové dotazy na 2 uzly s nedostupností, B-strom, 4 x 32

DDS v2.0 - Bodové dotazy na 5 uzlů					
Základní pohled	Virt_23				
B-strom					
	Čas [s]	**Dotázaných záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
Průměr	321,075	4 x 1 mil.	4 x 32	12 458	0
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	693 003	942 614	878 949	868 709	617 237
Dotazů	692 491	942 614	878 949	868 709	617 237
Čas DB [s]	5,345	6,269	5,725	6,26s	4,492
Max prop. [op/s]	2 749	3 384	3510	3 416	2 669
Průměr prop. [op/s]	2 158	2 738	2 737	2 705	1 922
Pohledů	0	128	128	128	128
Adresní chyby	512	0	0	0	0
R-strom					
	Čas [s]	**Dotázaných záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
Průměr	316,991	4 x 1 mil.	4 x 32	12 619	0
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	693 004	942 614	878 949	868 708	617 237
Dotazů	692 492	942 614	878 949	868 708	617 237
Čas DB [s]	127,23	130,556	134,03	131,213	90,381
Max prop. [op/s]	2 998	4 197	3 776	3 788	2 816
Průměr prop. [op/s]	2 186	2 974	2 773	2 740	1 947
Pohledů	0	128	128	128	128
Adresní chyby	512	0	0	0	0

Tabulka 30: DDS v2.0 - Bodové dotazy na 5 uzlů, B-strom a R-strom, 4 x 32

DDS v2.0 - Bodové dotazy na 10 uzlů					
Základní pohled	Virt_23 + Virt_28				
B-strom					
	Čas [s]	**Dotázaných záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
Průměr	256,547	4 x 1 mil.	4 x 32	15 592	0
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	520 909	702 446	654 832	629 058	502 973
Dotazů	520 525	702 446	654 832	629 058	502 973
Čas DB [s]	20,484	16,658	17,281	15,684	14,944
Max prop. [op/s]	2 649	3 697	3 343	3 367	2 622
Průměr prop. [op/s]	2 030	2 738	2 552	2 448	1 961
Pohledů	358 548	321 136	300 164	210 575	320 282
Adresní chyby	256	0	0	0	0
Uzel	**Virt_28**	**Virt_29**	**Virt_30**	**Virt_31**	**Virt_32**
Spojení celkem	172 095	240 168	224 117	239 650	114 364
Dotazů	171 967	240 168	224 117	239 650	114 364
Čas DB [s]	4,653	8,028	5,88	5,632	3,155
Max prop. [op/s]	1 333	1 764	1 993	1 911	1 235
Průměr prop. [op/s]	674	936	874	934	446
Pohledů	64 803	58 265	34 502	150 117	31 228
Adresní chyby	256	0	0	0	0
R-strom					
	Čas [s]	**Dotázaných záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
Průměr	250,385	4 x 1 mil.	4 x 32	15 975	0
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	583 653	941 883	534 580	753 403	256 968
Dotazů	583 397	941 883	534 580	753 403	256 968
Čas DB [s]	18,997	23,362	14,849	17,389	10,072
Max prop. [op/s]	2 949	4 217	3 673	3 620	1 844
Průměr prop. [op/s]	2 331	3 762	2 135	3 009	1 026
Pohledů	344 844	454 318	353 206	515 367	109 275
Adresní chyby	256	0	0	0	0
Uzel	**Virt_28**	**Virt_29**	**Virt_30**	**Virt_31**	**Virt_32**
Spojení celkem	109 351	731	344 370	115 305	360 269
Dotazů	108 905	731	344 370	115 305	360 269
Čas DB [s]	2,992	0,092	8,371	4,674	11,627
Max prop. [op/s]	1 935	65	3 206	1 971	1 760
Průměr prop. [op/s]	437	3	1 375	461	1 439
Pohledů	23 597	0	112 890	22 665	227 736
Adresní chyby	256	0	0	0	0

Tabulka 31: DDS v2.0 - Bodové dotazy na 10 uzlů, Bstrom a R-strom, 4 x 32

DDS v2.0 - Bodové dotazy na 15 uzlů					
Základní pohled	Virt_23 + Virt_28 + Virt_17				
B-strom					
	Čas [s]	**Dotázaných záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
Průměr	227,276	4 x 1 mil.	4 x 32	17 603	0
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	344 538	457 628	398 173	395 641	303 862
Dotazů	344 282	457 628	398 173	395 641	303 862
Čas DB [s]	10,59	11,193	9,533	9,782	8,981
Max prop. [op/s]	2 182	3 017	2 829	2 638	1 974
Průměr prop. [op/s]	1 516	2 014	1 752	1 741	1 337
Pohledů	123 096	247 949	111 537	191 401	92 854
Adresní chyby	256	0	0	0	0
Uzel	**Virt_28**	**Virt_29**	**Virt_30**	**Virt_31**	**Virt_32**
Spojení celkem	194 384	217 077	361 561	81 718	210 241
Dotazů	194 256	217 077	361 561	81 718	210 241
Čas DB [s]	5,373	5,381	9,192	2,527	5,905
Max prop. [op/s]	2 214	2 173	2 963	1 421	2 017
Průměr prop. [op/s]	855	955	1 591	360	925
Pohledů	101 674	131 333	232 074	23 095	148 214
Adresní chyby	128	0	0	0	0
	Virt_17	**Virt_40**	**Virt_95**	**Virt_96**	**Virt_97**
Spojení celkem	154 082	267 909	119 215	391 349	103 131
Dotazů	153 954	267 909	119 215	391 349	103 131
Čas DB [s]	8,383	11,024	3,174	15,859	3,197
Max prop. [op/s]	1 852	2 406	1 629	2 844	1 673
Průměr prop. [op/s]	678	1 179	525	1 721	454
Pohledů	107 507	63 784	27 024	214 496	18 386
Adresní chyby	128	0	0	0	0

Tabulka 32: DDS v2.0 - Bodové dotazy na 15 uzlů, B-strom, 4 x 32

DDS v2.0 - Bodové dotazy na 15 uzlů					
Základní pohled	Virt_23 + Virt_28 + Virt_17				
R-strom					
	Čas [s]	**Dotázaných záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
Průměr	235,461	4 x 1 mil.	4 x 32	17 071	0
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	221 816	444 280	12 362	187 638	309 042
Dotazů	221 560	444 280	12 362	187 638	309 042
Čas DB [s]	7,01	11,684	0,33	8,104	11,003
Max prop. [op/s]	1 502	2 509	540	2 306	1 682
Průměr prop. [op/s]	942	1 887	53	797	1 313
Pohledů	127 775	213 783	0	92 108	231 752
Adresní chyby	256	0	0	0	0
Uzel	**Virt_28**	**Virt_29**	**Virt_30**	**Virt_31**	**Virt_32**
Spojení celkem	290 705	461 594	544 309	212 266	445 181
Dotazů	290 554	461 594	544 309	212 266	445 181
Čas DB [s]	8,841	23,093	14,878	6,187	10,645
Max prop. [op/s]	2 382	2 470	2 744	1 451	3 708
Průměr prop. [op/s]	1 235	1 960	2 312	901	1 891
Pohledů	229 085	320 021	326 340	60 686	232 281
Adresní chyby	128	0	0	0	0
	Virt_17	**Virt_40**	**Virt_95**	**Virt_96**	**Virt_97**
Spojení celkem	180 516	36 748	33 774	324 406	95 933
Dotazů	180 388	36 748	33 774	324 406	95 933
Čas DB [s]	10,259	2,032	1,425	14,124	4,349
Max prop. [op/s]	1 454	1 243	1 503	2 549	769
Průměr prop. [op/s]	767	156	143	1 378	407
Pohledů	125 444	4 332	16 543	136 226	12 265
Adresní chyby	128	0	0	0	0

Tabulka 33: DDS v2.0 - Bodové dotazy na 15 uzlů, R-strom, 4 x 32

DDS v2.0 - Bodové dotazy na 20 uzlů					
Základní pohled	Virt_23 + Virt_28 + Virt_17 + Virt_98				
B-strom					
	Čas [s]	**Dotázaných záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
Průměr	213,228	4 x 1 mil.	4 x 32	18 769	0
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	238 510	255 998	215 274	272 427	77 399
Dotazů	238 382	255 998	215 274	272 427	77 399
Čas DB [s]	7,86	7,228	5,091	6,462	2,54
Max prop. [op/s]	1 957	1 745	1 462	1 965	1 198
Průměr prop. [op/s]	1 119	1 201	1 010	1 278	363
Pohledů	187 963	158 601	61 498	184 805	19 539
Adresní chyby	128	0	0	0	0
Uzel	**Virt_28**	**Virt_29**	**Virt_30**	**Virt_31**	**Virt_32**
Spojení celkem	187 398	304 563	150 861	206 945	403 003
Dotazů	187 270	304 563	150 861	206 945	403 003
Čas DB [s]	5,489	7,042	4,574	6,303	9,562
Max prop. [op/s]	1 707	2 899	938	1 278	2 262
Průměr prop. [op/s]	879	1 428	708	971	1 890
Pohledů	131 294	179 986	130 896	68 299	351 416
Adresní chyby	128	0	0	0	0
Uzel	**Virt_17**	**Virt_40**	**Virt_95**	**Virt_96**	**Virt_97**
Spojení celkem	180 225	153 826	49 555	207 239	225 813
Dotazů	180 097	153 826	49 555	207 239	225 813
Čas DB [s]	8,54	7,887	2,12	10,671	12,535
Max prop. [op/s]	1 110	1 299	1 960	1 611	1 640
Průměr prop. [op/s]	451	721	232	972	1 059
Pohledů	47 460	11 239	18 350	90 167	142 978
Adresní chyby	128	0	0	0	0
Uzel	**Virt_98**	**Virt_99**	**Virt_100**	**Virt_101**	**Virt_102**
Spojení celkem	170 871	228 227	211 117	128 097	133 164
Dotazů	170 745	228 227	211 117	128 097	133 164
Čas DB [s]	8,255	11,121	9,569	8,126	5,494
Max prop. [op/s]	1 324	1 551	1 473	1 283	1 093
Průměr prop. [op/s]	801	1 070	990	600	625
Pohledů	31 738	129 324	127 690	8 008	29 766
Adresní chyby	128	0	0	0	0

Tabulka 34: DDS v2.0 - Bodové dotazy na 20 uzlů, B-strom, 4 x 32

DDS v2.0 - Bodové dotazy na 20 uzlů					
Základní pohled	Virt_23 + Virt_28 + Virt_17 + Virt_98				
R-strom					
	Čas [s]	**Dotázaných záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
Průměr	208,346	4 x 1 mil.	4 x 32	19 239	0
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	137 026	169 518	218 488	323 264	22 276
Dotazů	136 898	169 518	218 488	323 264	22 276
Čas DB [s]	4,201	7,64	5,734	9,228	0,763
Max prop. [op/s]	1 004	1 956	1 618	2 277	843
Průměr prop. [op/s]	658	813	1 049	1 552	107
Pohledů	97 567	210 007	163 333	198 188	2 362
Adresní chyby	128	0	0	0	0
Uzel	**Virt_28**	**Virt_29**	**Virt_30**	**Virt_31**	**Virt_32**
Spojení celkem	176 659	259 934	93 030	188 034	54 864
Dotazů	176 531	259 934	93 030	188 034	54 864
Čas DB [s]	4,331	6,104	2,471	4,174	1,176
Max prop. [op/s]	1 415	2 131	1 562	1 489	1 060
Průměr prop. [op/s]	847	1 248	446	902	263
Pohledů	46 118	126 991	18 460	54 421	20 876
Adresní chyby	128	0	0	0	0
Uzel	**Virt_17**	**Virt_40**	**Virt_95**	**Virt_96**	**Virt_97**
Spojení celkem	56 318	189 241	349 811	176 782	401 482
Dotazů	56 190	189 241	349 811	176 782	401 482
Čas DB [s]	5,607	9,328	20,603	8,405	23,9
Max prop. [op/s]	897	1 371	2 963	1 910	2 487
Průměr prop. [op/s]	270	908	1 679	849	1 927
Pohledů	15 849	19 976	144 945	67 368	366 122
Adresní chyby	128	0	0	0	0
Uzel	**Virt_98**	**Virt_99**	**Virt_100**	**Virt_101**	**Virt_102**
Spojení celkem	323 001	223 921	217 620	180 628	138 615
Dotazů	322 873	223 921	217 620	180 628	138 615
Čas DB [s]	17,006	12,981	12,012	8,067	7,877
Max prop. [op/s]	2 082	1 512	1 424	1 354	933
Průměr prop. [op/s]	1 550	1 075	1 045	867	665
Pohledů	306 031	130 413	192 520	133 864	33 353
Adresní chyby	128	0	0	0	0

Tabulka 35: DDS v2.0 - Bodové dotazy na 20 uzlů, R-strom, 4 x 32

DDS v2.0 - Bodové dotazy na 5 uzlů					
Základní pohled	Virt_23				
B-strom					
	Čas [s]	**Dotázaných záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
Průměr	588,441	8 x 1 mil.	8 x 32	13 596	2 890
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	1 386 009	1 883 919	1 757 130	1 737 264	1 234 474
Dotazů	1 384 985	1 883 919	1 757 130	1 737 264	1 234 474
Čas DB [s]	32,033	40,108	42,271	36,434	28,935
Max prop. [op/s]	3 169	4 255	4 011	3 927	2 825
Průměr prop. [op/s]	2 355	3 202	2 986	2 952	2 098
Pohledů	0	256	256	256	256
Adresní chyby	1024	0	0	0	0
R-strom					
	Čas [s]	**Dotázaných záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
Průměr	443,153	8 x 1 mil.	8 x 32	18 055	2 200
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	1 386 008	1 885 203	1 757 866	1 737 416	1 234 323
Dotazů	1 384 984	1 885 203	1 757 866	1 737 416	1 234 323
Čas DB [s]	40,97	47,06	42,933	35,708	25,42
Max prop. [op/s]	4 183	5 468	4 940	4 939	4 201
Průměr prop. [op/s]	3 128	4 254	3 967	3 920	2 785
Pohledů	0	256	256	256	256
Adresní chyby	1024	0	0	0	0

Tabulka 36: DDS v2.0 - Bodové dotazy na 5 uzlů, B-strom a R-strom, 4 x 32

DDS v2.0 - Bodové dotazy na 10 uzlů					
Základní pohled	Virt_23 + Virt_28				
B-strom					
	Čas [s]	**Dotázaných záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
Průměr	344,538	8 x 1 mil.	8 x 32	23 226	1 704
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	637 334	687 047	1 279 140	1 462 223	538 293
Dotazů	637 822	687 047	1 279 140	1 462 223	538 293
Čas DB [s]	19,161	16,799	37,158	31,211	13,678
Max prop. [op/s]	2 788	4 153	5 801	4 934	2 604
Průměr prop. [op/s]	1 850	1 994	3 713	4 244	1 562
Pohledů	314 700	362 581	1 019 413	893 838	167 423
Adresní chyby	512	0	0	0	0
Uzel	**Virt_28**	**Virt_29**	**Virt_30**	**Virt_31**	**Virt_32**
Spojení celkem	747 674	1 198 181	487 758	275 193	686 981
Dotazů	747 162	1 198 181	487 758	275 193	686 981
Čas DB [s]	17,429	24,877	11,533	7,228	14,255
Max prop. [op/s]	3 442	5 195	5 635	3 232	3 391
Průměr prop. [op/s]	2 170	3 478	1 416	799	1 994
Pohledů	581 745	672 302	312 150	114 875	249 497
Adresní chyby	512	0	0	0	0
R-strom					
	Čas [s]	**Dotázaných záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
Průměr	343,459	8 x 1 mil.	8 x 32	23 327	2 409
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	393 456	1 206 557	488 988	622 257	1 230 520
Dotazů	392 944	1 206 557	488 988	622 257	1 230 520
Čas DB [s]	80,765	129,522	106,65	116,126	109,019
Max prop. [op/s]	4 277	5 582	3 564	4 090	4 387
Průměr prop. [op/s]	1 146	3 513	1 424	1 812	3 583
Pohledů	243 698	653 905	344 113	206 214	864 792
Adresní chyby	512	0	0	0	0
Uzel	**Virt_28**	**Virt_29**	**Virt_30**	**Virt_31**	**Virt_32**
Spojení celkem	992 552	678 622	1 268 717	1 115 154	3 954
Dotazů	992 040	678 622	1 268 717	1 115 154	3 954
Čas DB [s]	20,251	35,253	29,972	29,069	0,22
Max prop. [op/s]	4 402	3 989	4 885	5 358	80
Průměr prop. [op/s]	2 890	1 976	3 694	3 247	12
Pohledů	806 968	230 980	896 329	731 969	0
Adresní chyby	512	0	0	0	0

Tabulka 37: DDS v2.0 - Bodové dotazy na 10 uzlů, R-strom a B-strom, 8 x 32

DDS v2.0 - Bodové dotazy na 10 uzlů - chyba					
Základní pohled	Virt_23 + Virt_28				
B-strom					
	Čas [s]	**Dotázaných záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
Průměr	627,503	8 x 1 mil.	8 x 32	12 754	827
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	321 097	716 544	515 158	1 338 591	531 614
Dotazů	320 585	716 544	515 158	1 338 591	531 614
Čas DB [s]	17,564	29,364	19,975	47,303	21,228
Max prop. [op/s]	2 819	3 107	3 237	3 532	2 493
Průměr prop. [op/s]	512	1 142	821	2 133	847
Pohledů	372 304	250 724	985 036	348 763	243 993
Adresní chyby	512	0	0	0	0
Uzel	**Virt_28**	**Virt_29**	**Virt_30**	**Virt_31**	**Virt_32**
Spojení celkem	1 064 911	1 168 674	1 242 697	398 812	702 860
Dotazů	1 064 399	1 168 674	1 242 697	398 812	702 860
Čas DB [s]	28,074	39,637	35,908	10,873	21,264
Max prop. [op/s]	2 961	3 488	3 529	3 424	2 663
Průměr prop. [op/s]	1 697	1 862	1 980	636	1 120
Pohledů	704 162	668 386	919 564	158 343	531 015
Adresní chyby	512	0	0	0	0

Tabulka 38: DDS v2.0 - Bodové dotazy na 10 uzlů, B-strom, 8 x 32 - chyba

DDS v2.0 - Bodové dotazy na 15 uzlů					
Základní pohled	Virt_23 + Virt_28 + Virt_17				
B-strom					
	Čas [s]	**Dotázaných záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
Průměr	353,077	8 x 1 mil.	8 x 32	22 693	0
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	322 558	450 072	337 355	308 399	291 156
Dotazů	322 302	450 072	337 355	308 399	291 156
Čas DB [s]	5,657	7,987	5,858	6,869	6,862
Max prop. [op/s]	1 929	2 584	2 334	1 774	1 610
Průměr prop. [op/s]	913	1 275	955	874	825
Pohledů	144 238	386 625	147 733	100 377	99 545
Adresní chyby	256	0	0	0	0
Uzel	**Virt_28**	**Virt_29**	**Virt_30**	**Virt_31**	**Virt_32**
Spojení celkem	604 416	930 338	647 487	712 859	179 807
Dotazů	604 032	930 338	647 487	712 859	179 807
Čas DB [s]	11,866	17,906	12,159	13,912	3,522
Max prop. [op/s]	2 291	4 305	3 295	3 410	1 656
Průměr prop. [op/s]	1 712	2 635	1 834	2 019	509
Pohledů	318 888	630 947	350 596	340 003	64 044
Adresní chyby	384	0	0	0	0
Uzel	**Virt_17**	**Virt_40**	**Virt_95**	**Virt_96**	**Virt_97**
Spojení celkem	457 157	502 781	772 361	717 335	766 953
Dotazů	456 763	502 781	772 361	717 335	766 953
Čas DB [s]	24,866	31,075	28,548	25,633	46,091
Max prop. [op/s]	1 781	2 790	3 422	3 863	3 225
Průměr prop. [op/s]	1 295	1 424	2 188	2 032	2 172
Pohledů	242 713	38 738	429 306	199 543	519 919
Adresní chyby	384	0	0	0	0

Tabulka 39: DDS v2.0 - Bodové dotazy na 15 uzlů, B-strom, 8 x 32

DDS v2.0 - Bodové dotazy na 15 uzlů					
Základní pohled	Virt_23 + Virt_28 + Virt_17				
R-strom					
	Čas [s]	**Dotázaných záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
Průměr	359,979	8 x 1 mil.	8 x 32	22 311	0
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	102 601	199 850	0	0	151 046
Dotazů	102 729	199 850	0	0	151 046
Čas DB [s]	34,016	32,896	0	0	21,263
Max prop. [op/s]	850	1 161	0	0	916
Průměr prop. [op/s]	285	555	0	0	420
Pohledů	67 947	114 391	0	0	99 940
Adresní chyby	128	0	0	0	0
Uzel	**Virt_28**	**Virt_29**	**Virt_30**	**Virt_31**	**Virt_32**
Spojení celkem	1 021 751	1 479 103	512 016	1 260 165	858 726
Dotazů	1 021 239	1 479 103	512 016	1 260 165	858 726
Čas DB [s]	21,77	31,569	9,037	102,442	17,741
Max prop. [op/s]	3 719	5 230	3 197	4 467	4 061
Průměr prop. [op/s]	2 838	4 109	1 422	3 500	2 386
Pohledů	805 223	1 113 897	255 926	902 860	611 094
Adresní chyby	512	0	0	0	0
Uzel	**Virt_17**	**Virt_40**	**Virt_95**	**Virt_96**	**Virt_97**
Spojení celkem	261 460	36 748	1 245 882	477 248	224 702
Dotazů	261 061	36 748	1 245 882	477 248	224 702
Čas DB [s]	14,393	2,032	53,663	16,855	15,638
Max prop. [op/s]	1 932	1 243	4 912	5 149	1 774
Průměr prop. [op/s]	726	102	3 461	1 326	624
Pohledů	123 226	4 332	1 124 117	275 742	29 372
Adresní chyby	384	0	0	0	0

Tabulka 40: DDS v2.0 - Bodové dotazy na 15 uzlů, R-strom, 8 x 32

DDS v2.0 - Bodové dotazy na 20 uzlů					
Základní pohled	Virt_23 + Virt_28 + Virt_17 + Virt_98				
B-strom					
	Čas [s]	**Dotázaných záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
Průměr	424,814	8 x 1 mil.	8 x 32	19 513	0
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	42 789	669 822	355 894	639 907	226 189
Dotazů	42 533	669 822	355 894	639 907	226 189
Čas DB [s]	4,842	16,466	7,328	14,997	7,601
Max prop. [op/s]	1 105	3 631	1 915	2 637	1 556
Průměr prop. [op/s]	101	1 577	838	1 506	532
Pohledů	7 590	366 599	147 325	459 866	137 165
Adresní chyby	256	0	0	0	0
Uzel	**Virt_28**	**Virt_29**	**Virt_30**	**Virt_31**	**Virt_32**
Spojení celkem	517 313	550 340	526 357	388 522	300 426
Dotazů	517 057	550 340	526 357	388 522	300 426
Čas DB [s]	10,473	10,843	9,941	9,963	6,148
Max prop. [op/s]	2 264	3 384	3 017	2 652	2 025
Průměr prop. [op/s]	1 218	1 296	1 239	915	707
Pohledů	389 626	287 092	401 512	140 227	225 962
Adresní chyby	256	0	0	0	0
Uzel	**Virt_17**	**Virt_40**	**Virt_95**	**Virt_96**	**Virt_97**
Spojení celkem	225 924	348 245	693 913	385 973	379 634
Dotazů	225 668	348 245	693 913	385 973	379 634
Čas DB [s]	20,739	24,129	34,778	22,979	25,442
Max prop. [op/s]	1 193	1 497	2 061	1 543	1 774
Průměr prop. [op/s]	425	820	1 634	909	894
Pohledů	124 995	12 606	438 021	204 623	221 665
Adresní chyby	256	0	0	0	0
Uzel	**Virt_98**	**Virt_99**	**Virt_100**	**Virt_101**	**Virt_102**
Spojení celkem	599 982	316 821	412 720	323 016	288 225
Dotazů	599 726	316 821	412 720	323 016	288 225
Čas DB [s]	35,042	16,884	15,725	20,49	17,398
Max prop. [op/s]	2 835	1 712	1783	1 599	1 209
Průměr prop. [op/s]	1 412	746	972	760	679
Pohledů	534 173	42 845	387 086	45 758	219 336
Adresní chyby	256	0	0	0	0

Tabulka 41: DDS v2.0 - Bodové dotazy na 20 uzlů, B-strom, 8 x 32

DDS v2.0 - Bodové dotazy na 20 uzlů					
Základní pohled	Virt_23 + Virt_28 + Virt_17 + Virt_98				
R-strom					
	Čas [s]	**Dotázaných záznamů**	**Simulovaných klientů**	**Propustnost [op/s]**	**Chyb přenosu**
Průměr	399,880	8 x 1 mil.	8 x 32	20 415	0
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	99 100	556 785	261 955	443 438	498 125
Dotazů	98 844	556 785	261 955	443 438	498 125
Čas DB [s]	64,962	105,013	6,051	114,581	75,264
Max prop. [op/s]	633	3 132	2 367	2 933	3 399
Průměr prop. [op/s]	248	1 392	655	1 108	1 246
Pohledů	17 013	209 548	149 324	163 876	324 315
Adresní chyby	256	0	0	0	0
Uzel	**Virt_28**	**Virt_29**	**Virt_30**	**Virt_31**	**Virt_32**
Spojení celkem	455 530	516 306	695 387	424 954	364 355
Dotazů	455 274	516 306	695 387	424 954	364 355
Čas DB [s]	9,026	11,048	14,748	75,075	10,235
Max prop. [op/s]	2 679	2 991	3 872	2 498	1 955
Průměr prop. [op/s]	1 139	1 291	1 739	1 063	911
Pohledů	346 903	140 812	501 796	192 046	284 927
Adresní chyby	256	0	0	0	0
Uzel	**Virt_17**	**Virt_40**	**Virt_95**	**Virt_96**	**Virt_97**
Spojení celkem	163 028	433 196	395 944	434 630	201 661
Dotazů	162 772	433 196	395 944	434 630	201 661
Čas DB [s]	14,626	30,55	24,928	24,879	19,008
Max prop. [op/s]	1 264	1 662	1 572	1 616	1 169
Průměr prop. [op/s]	408	1 083	990	1 087	504
Pohledů	64 826	68 770	113 166	128 956	56 072
Adresní chyby	256	0	0	0	0
Uzel	**Virt_98**	**Virt_99**	**Virt_100**	**Virt_101**	**Virt_102**
Spojení celkem	668 094	378 942	404 612	434 394	201 661
Dotazů667 838		378 942	404 612	434 394	201 661
Čas DB [s]	38,001	22,565	30,339	27,469	14,386
Max prop. [op/s]	2 849	1 626	1 699	1 751	1 147
Průměr prop. [op/s]	1 670	948	1 012	1 086	1 169
Pohledů	644 805	253 112	109 229	390 085	8 054
Adresní chyby	256	0	0	0	0

Tabulka 42: DDS v2.0 - Bodové dotazy na 20 uzlů, R-strom, 8 x 32

D Tabulky rozsahových dotazů

DDS v2.0 - Rozsahové dotazy na 5 uzlů					
Základní pohled	Virt_23				
Dotazů	1 mil.	**Klientů**	1 x 32	**Chyb**	0

B-strom					
	Čas [s]	**Propustnost [op/s]**	**Výsledků celkem**	**Max výsledků v 1 odpovědi**	**Výsledků [s]**
Průměr	267,890	3 733	204 874 723	1 225	764 827
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	802 298	833 791	821 619	844 550	820 925
Dotazů	802 170	833 791	821 619	844 550	820 925
Replikací dotazu	547 176	682 768	740 621	672 714	479 660
Čas DB [s]	21,815	32,992	31,972	32,022	20,356
Max prop. [op/s]	3 393	3 515	3 492	3 578	3 474
Průměr prop. [op/s]	2 995	3 112	3 067	3 153	3 064
Pohledů	0	32	32	32	32
Adresní chyby	128	0	0	0	0

R-strom					
	Čas [s]	**Propustnost [op/s]**	**Výsledků celkem**	**Max výsledků v 1 odpovědi**	**Výsledků [s]**
Průměr	3 108,846	322	6 270 839	89	2 017
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	805 995	824 238	836 432	846 538	822 946
Dotazů	805 986	824 238	836 432	846 538	822 946
Replikací dotazu	553 822	738 350	690 638	674 490	478 760
Čas DB [s]	2 128,95	2 792,37	2 097,54	2 920,43	2 151,59
Max prop. [op/s]	2 484	2 143	2 515	2 618	2 564
Průměr prop. [op/s]	259	265	269	269	265
Pohledů	0	32	32	32	32
Adresní chyby	128	0	0	0	0

Tabulka 43: DDS v2.0 - Rozsahové dotazy na 5 uzlů, B-strom a R-strom, 32

DDS v2.0 - Rozsahové dotazy na 10 uzlů, B-strom					
Základní pohled	Virt_23 + Virt_28				
Dotazů	1 mil.	**Klientů**	32	**Chyb**	0
	Čas [s]	**Propustnost [op/s]**	**Výsledků celkem**	**Max výsledků v 1 odpovědi**	**Výsledků [s]**
Průměr	243,154	4 113	205 281 345	1 042	845 461
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	457 312	498 976	536 374	527 588	500 824
Dotazů	457 248	498 976	536 374	527 588	500 824
Replikací dotazu	272 683	626 789	374 985	332 811	235 541
Čas DB [s]	19,227	25,11	26,079	26,043	20,488
Max prop. [op/s]	2 095	2 335	2 484	2 408	2 297
Průměr prop. [op/s]	1 880	2 050	2 205	2 170	2 060
Pohledů	13 726	98 702	66 661	80 873	51 924
Adresní chyby	64	0	0	0	0
Uzel	**Virt_28**	**Virt_29**	**Virt_30**	**Virt_31**	**Virt_32**
Spojení celkem	341 807	319 583	294 230	313 574	315 608
Dotazů	341 743	319 583	294 230	313 574	315 608
Replikací dotazu	272 983	104 743	309 840	335 891	392 220
Čas DB [s]	12,887	15,017	13,711	15,544	13,158
Max prop. [op/s]	1 689	1 642	1 510	1 567	1 621
Průměr prop. [op/s]	1 406	1 314	1 210	1 290	1 298
Pohledů	14 731	25 899	32 291	54 021	43 550
Adresní chyby	64	0	0	0	0

Tabulka 44: DDS v2.0 - Rozsahové dotazy na 10 uzlů, B-strom, 32

DDS v2.0 - Rozsahové dotazy na 10 uzlů					
Základní pohled	Virt_23 + Virt_28				
Dotazů	1 mil.	**Klientů**	32	**Chyb**	0
R-strom					
	Čas [s]	**Propustnost [op/s]**	**Výsledků celkem**	**Max výsledků v 1 odpovědi**	**Výsledků [s]**
Průměr	5 137,033	195	6 402 464	95	1 301
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	579 989	335 158	352 414	529 885	381 978
Dotazů	579 925	335 158	352 414	529 885	381 978
Replikací dotazu	176 133	336 779	234 468	414 216	381 978
Čas DB [s]	4 351,41	2 617,92	1 052,41	4 863,05	2 494,04
Max prop. [op/s]	1 173	1 070	1 164	1 205	141
Průměr prop. [op/s]	112	65	69	103	74
Pohledů	176 133	41	58	289	141
Adresní chyby	64	0	0	0	0
Uzel	**Virt_28**	**Virt_29**	**Virt_30**	**Virt_31**	**Virt_32**
Spojení celkem	223 383	487 430	477 078	315 314	439 184
Dotazů	223 319	487 430	477 078	315 314	439 184
Replikací dotazu	373 686	399 645	454 779	259 553	63 500
Čas DB [s]	1 365,18	5 540,3	5 847,36	3 640	4 518,19
Max prop. [op/s]	68	174	160	143	256
Průměr prop. [op/s]	43	95	92	61	85
Pohledů	0	16	16	16	16
Adresní chyby	64	0	0	0	0

Tabulka 45: DDS v2.0 - Rozsahové dotazy na 10 uzlů, R-strom, 32

DDS v2.0 - Rozsahové dotazy na 10 uzlů					
Základní pohled	Virt_23 + Virt_28				
Dotazů	2 x 1 mil.	**Klientů**	2 x 32	**Chyb**	0
B-strom					
	Čas [s]	**Propustnost [op/s]**	**Výsledků celkem**	**Max výsledků v 1 odpovědi**	**Výsledků [s]**
Průměr	416,296	4 818	403 839 628	1 070	972 492
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	890 841	908 527	1 188 621	1 131 624	1 074 424
Dotazů	890 713	908 527	1 188 621	1 131 624	1 074 424
Replikací dotazu	556 134	815 725	1 192 450	434 524	533 534
Čas DB [s]	36,223	45,074	55,226	53,341	40,994
Max prop. [op/s]	2 547	2 592	3 385	3 365	3 190
Průměr prop. [op/s]	2 140	2 182	2 855	2 718	2 581
Pohledů	82 610	142 867	280 926	44 252	53 849
Adresní chyby	128	0	0	0	0
Uzel	**Virt_28**	**Virt_29**	**Virt_30**	**Virt_31**	**Virt_32**
Spojení celkem	715 456	736 553	480 003	557 492	567 399
Dotazů	715 328	736 553	480 003	557 492	567 399
Replikací dotazu	543 508	661 055	179 139	910 386	423 458
Čas DB [s]	29,06	36,774	21,136	29,093	23,22
Max prop. [op/s]	2 672	2 694	2 026	2 289	2 256
Průměr prop. [op/s]	1 718	1 769	1 153	1 339	1 363
Pohledů	59 433	139 505	35 651	150 678	70 704
Adresní chyby	128	0	0	0	0

Tabulka 46: DDS v2.0 - Rozsahové dotazy na 10 uzlů, B-strom, 2 x 32

DDS v2.0 - Rozsahové dotazy na 15 uzlů					
Základní pohled	Virt_23 + Virt_28 + Virt_17				
Dotazů	1 mil.	**Klientů**	32	**Chyb**	0
B-strom					
	Čas [s]	**Propustnost [op/s]**	**Výsledků celkem**	**Max výsledků v 1 odpovědi**	**Výsledků [s]**
Průměr	290,522	3 442	201 674 410	1 016	695 660
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	325 964	324 823	321 872	269 716	260 348
Dotazů	325 924	324 823	321 872	269 716	260 348
Replikací dotazu	325 964	242 724	226 328	201 991	144 572
Čas DB [s]	13,925	17,212	16,347	11,463	8,754
Max prop. [op/s]	1 723	1 695	1 676	1 476	1 464
Průměr prop. [op/s]	1 122	1 118	1 108	928	896
Pohledů	46 090	67 128	21 970	43 270	12 375
Adresní chyby	40	0	0	0	0
Uzel	**Virt_28**	**Virt_29**	**Virt_30**	**Virt_31**	**Virt_32**
Spojení celkem	250 360	253 693	258 979	269 737	255 757
Dotazů	250 320	253 693	258 979	269 737	255 757
Replikací dotazu	174 775	214 530	228 682	252 719	104 579
Čas DB [s]	8,738	14,759	11,177	11,93	9,108
Max prop. [op/s]	1 765	1 781	1 754	1 749	1 659
Průměr prop. [op/s]	861	873	891	928	880
Pohledů	16 610	58 767	29 723	29 841	23 035
Adresní chyby	40	0	0	0	0
Uzel	**Virt_17**	**Virt_40**	**Virt_95**	**Virt_96**	**Virt_97**
Spojení celkem	226 895	242 053	251 536	303 924	303 501
Dotazů	226 847	242 053	251 536	303 924	303 501
Replikací dotazu	192 649	274 160	234 032	214 867	230 774
Čas DB [s]	17,757	14,125	24,356	27,771	19,074
Max prop. [op/s]	1 956	1 141	1 166	1 380	1 440
Průměr prop. [op/s]	781	833	866	1 046	1 045
Pohledů	26 286	85 278	31 247	30 236	70 955
Adresní chyby	48	0	0	0	0

Tabulka 47: DDS v2.0 - Rozsahové dotazy na 15 uzlů, B-strom, 32

DDS v2.0 - Rozsahové dotazy na 15 uzlů					
Základní pohled	Virt_23 + Virt_28 + Virt_17				
Dotazů	1 mil.	**Klientů**	32	**Chyb**	0
R-strom					
	Čas [s]	**Propustnost [op/s]**	**Výsledků celkem**	**Max výsledků v 1 odpovědi**	**Výsledků [s]**
Průměr	5 652,437	177	5 969 907	88	1 069
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	153 849	175 718	109 925	95 846	142 874
Dotazů	153 809	175 718	109 925	95 846	142 874
Replikací dotazu	184 404	241 455	0	0	158 161
Čas DB [s]	320,486	455,596	343,455	343,357	253,069
Max prop. [op/s]	123	128	89	78	130
Průměr prop. [op/s]	27	31	19	17	25
Pohledů	0	10	10	10	10
Adresní chyby	40	0	0	0	0
Uzel	**Virt_28**	**Virt_29**	**Virt_30**	**Virt_31**	**Virt_32**
Spojení celkem	381 376	371 231	445 887	467 110	403 710
Dotazů	381 376	371 231	445 887	467 110	403 710
Replikací dotazu	183 136	247 119	460 991	447 425	158 113
Čas DB [s]	3140,98	3188,75	4233,45	4424,66	3196,73
Max prop. [op/s]	146	137	104	192	149
Průměr prop. [op/s]	67	66	79	83	71
Pohledů	0	10	10	10	10
Adresní chyby	40	0	0	0	0
Uzel	**Virt_17**	**Virt_40**	**Virt_95**	**Virt_96**	**Virt_97**
Spojení celkem	267 997	273 615	277 134	281 162	273 879
Dotazů	267 949	273 615	277 134	281 162	273 879
Replikací dotazu	182 027	242 972	228 309	223 104	163 986
Čas DB [s]	5 362,31	463,1	5 167,29	5 593,44	5729,87
Max prop. [op/s]	73	76	81	81	74
Průměr prop. [op/s]	47	48	49	50	49
Pohledů	0	12	12	12	12
Adresní chyby	48	0	0	0	0

Tabulka 48: DDS v2.0 - Rozsahové dotazy na 15 uzlů, R-strom, 32

DDS v2.0 - Rozsahové dotazy na 15 uzlů					
Základní pohled	Virt_23 + Virt_28 + Virt_17				
Dotazů	2 x 1 mil.	**Klientů**	2 x 32	**Chyb**	0
B-strom					
	Čas [s]	**Propustnost [op/s]**	**Výsledků celkem**	**Max výsledků v 1 odpovědi**	**Výsledků [s]**
Průměr	358,162	5 597	399 149 789	977	1 117 001
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	543 185	545 495	548 005	476 212	468 376
Dotazů	543 101	545 495	548 005	476 212	468 376
Replikací dotazu	208 886	447 907	430 002	363 688	304 980
Čas DB [s]	22,931	26,729	25,528	20,078	19,662
Max prop. [op/s]	2 102	2 076	2 112	2 061	1 978
Průměr prop. [op/s]	1 517	1 523	1 530	1 330	1 308
Pohledů	34 426	105 067	115 694	56 885	40 695
Adresní chyby	84	0	0	0	0
Uzel	**Virt_28**	**Virt_29**	**Virt_30**	**Virt_31**	**Virt_32**
Spojení celkem	608 910	620 832	631 557	625 291	601 381
Dotazů	608 822	620 832	631 557	625 291	601 381
Replikací dotazu	544 780	524 018	525 372	414 031	331 497
Čas DB [s]	25,935	31,021	27,758	31,558	24,392
Max prop. [op/s]	2 477	2 503	2 507	2 511	2 471
Průměr prop. [op/s]	1 700	1 734	1 764	1 746	1 679
Pohledů	142 232	49 558	112 089	77 838	23 454
Adresní chyby	88	0	0	0	0
Uzel	**Virt_17**	**Virt_40**	**Virt_95**	**Virt_96**	**Virt_97**
Spojení celkem	454 702	477 257	487 335	586 551	571 773
Dotazů	454 618	477 257	487 335	586 551	571 773
Replikací dotazu	342 340	495 211	416 393	569 647	327 095
Čas DB [s]	48,874	34,802	41,599	55,997	31,031
Max prop. [op/s]	2 114	2 169	2 216	2 459	2 509
Průměr prop. [op/s]	1 270	1 333	1 361	1 638	1 597
Pohledů	52 237	81 854	15 291	141 187	57 443
Adresní chyby	84	0	0	0	0

Tabulka 49: DDS v2.0 - Rozsahové dotazy na 15 uzlů, B-strom, 2 x 32

DDS v2.0 - Rozsahové dotazy na 20 uzlů					
Základní pohled	Virt_23 + Virt_28 + Virt_17 + Virt_98				
Dotazů	4 mil.	**Klientů**	2 x 32	**Nedostupnost**	7
B-strom					
	Čas [s]	**Propustnost [op/s]**	**Výsledků celkem**	**Max výsledků v 1 odpovědi**	**Výsledků [s]**
Průměr	2 198,210	1 820	49 666 3544	111	225 940
Uzel	**Virt_23**	**Virt_24**	**Virt_25**	**Virt_26**	**Virt_27**
Spojení celkem	728 473	801 946	814 592	866 068	866 963
Dotazů	728 409	801 946	814 592	866 068	866 963
Replikací dotazu	197 531	903 724	680 329	926 595	606 401
Čas DB [s]	23,175	32,898	29,889	32,898	27,116
Max prop. [op/s]	2 206	2 124	2 181	2 319	2 326
Průměr prop. [op/s]	331	365	371	394	394
Pohledů	32 311	145 384	113 526	132 626	96 970
Adresní chyby	64	0	0	0	0
Uzel	**Virt_28**	**Virt_29**	**Virt_30**	**Virt_31**	**Virt_32**
Spojení celkem	1 395 345	778 298	818 446	831 610	796 081
Dotazů	1 395 281	778 298	818 446	831 610	796 081
Replikací dotazu	0	947 226	808 935	581 431	449 558
Čas DB [s]	42,851	30,755	27,542	29,047	24,282
Max prop. [op/s]	2 999	2 270	2 383	2 401	2 386
Průměr prop. [op/s]	635	354	372	378	362
Pohledů	213 857	141 989	147 101	115 549	76 801
Adresní chyby	64	0	0	0	0
Uzel	**Virt_17**	**Virt_40**	**Virt_95**	**Virt_96**	**Virt_97**
Spojení celkem	650 585	656 849	684 504	725 371	732 696
Dotazů	650 521	656 849	684 504	725 371	732 696
Replikací dotazu	3 871	619 100	681 949	748 878	556 502
Čas DB [s]	107,031	99,299	82,322	90,694	85,56
Max prop. [op/s]	602	625	701	671	670
Průměr prop. [op/s]	296	298	311	330	333
Pohledů	97	94 929	138 279	140 002	84 239
Adresní chyby	64	0	0	0	0
Uzel	**Virt_98**	**Virt_99**	**Virt_100**	**Virt_101**	**Virt_102**
Spojení celkem	437 778	448 471	484 033	501 497	495 211
Dotazů	437 714	448 471	484 033	501 497	495 211
Replikací dotazu	5 197	463 186	568 362	435 601	318 070
Čas DB [s]	30,176	32,933	28,285	30,649	26,684
Max prop. [op/s]	570	637	757	708	710
Průměr prop. [op/s]	199	204	220	228	225
Pohledů	149	27 311	76 257	32 493	57 996
Adresní chyby	64	0	0	0	0

Tabulka 50: DDS v2.0 - Rozsahové dotazy na 20 uzlů, B-strom,2 x 32

E Testy síťové komunikace

C1xS1 - Test LAN 100xConnect-SEND-RECV-FIN										
	Vláken serveru:			1	Klientů:		1	Počet spojení:		100
	Velikost zprávy / čas testu [s]									
Test č.	0,5 MB	1 MB	2 MB	3 MB	4 MB	6 MB	8 MB	10 MB	12 MB	14 MB
1	5,382	10,624	21,200	31,622	41,933	62,681	83,850	104,254	125,268	146,157
2	5,367	10,609	21,158	31,590	41,855	62,853	83,475	104,177	125,175	146,437
3	5,428	10,545	21,158	31,574	41,964	62,806	83,445	104,224	125,221	146,094
4	5,429	10,608	21,201	31,637	41,870	62,853	83,382	104,287	125,377	146,001
5	5,304	10,561	21,169	31,527	41,854	62,884	83,741	104,661	125,440	146,016
6	5,304	10,624	21,170	31,606	41,855	62,884	83,803	105,472	125,284	145,969
7	5,304	10,576	21,170	31,606	41,933	62,805	83,616	104,988	125,299	145,891
8	5,538	10,530	21,200	31,653	41,979	62,962	83,398	105,129	125,393	145,876
9	5,320	10,592	21,154	31,574	42,058	62,899	83,413	104,427	125,253	146,063
10	5,288	10,655	21,169	31,512	41,652	62,852	83,398	104,271	125,222	146,188
11	5,304	10,545	21,185	31,419	42,260	62,868	83,491	104,114	125,143	146,172
12	5,320	10,624	21,138	31,496	42,152	62,868	83,460	104,239	125,330	146,047
13	5,531	10,545	21,154	31,45	42,042	62,884	83,413	104,146	125,362	146,016
14	5,304	10,577	21,184	31,449	42,213	62,712	83,351	104,223	125,159	146,328
15	5,320	10,561	21,185	31,512	42,089	62,899	83,257	104,193	125,237	145,892
16	5,351	10,499	21,216	31,466	42,058	62,837	83,695	104,223	125,175	146,126
17	5,304	10,530	21,185	31,481	42,167	62,852	83,429	104,271	125,128	145,876
18	5,319	10,515	21,154	31,512	42,042	62,962	83,382	104,146	125,034	146,187
19	5,367	10,608	21,123	31,480	42,136	62,946	83,257	104,208	125,253	146,344
20	5,382	10,592	21,153	31,419	42,011	62,806	83,476	104,426	125,268	146,173
Průměr [s]	5,358	10,576	21,171	31,529	42,01	62,855	83,486	104,404	125,251	146,093
Spojení celkem	19	9	5	3	2	2	1	1	1	1
Propustnost [MB/s]	9,332	9,455	9,446	9,514	9,522	9,545	9,582	9,578	9,581	9,583

Tabulka 51: C1xS1 - Test LAN 100xConnect-SEND-RECV-FIN

C1xS1 - Test LAN 100xConnect-SEND-RECV-FIN												
	Vláken serveru:			1		Klientů:		1		Počet spojení:		100
	Velikost zprávy / čas testu [s]											
Test č.	**100 B**	**500 B**	**1260 B**	**1,5 kB**	**2kB**	**4 kB**	**8 kB**	**16 kB**	**32 kB**	**64 kB**	**128 kB**	**256 kB**
1	0,811	0,842	0,983	1,030	1,045	1,622	2,153	3,058	4,852	8,097	14,789	27,284
2	0,797	0,905	0,967	1,310	1,061	1,669	2,137	3,104	5,523	8,112	14,555	27,503
3	0,577	0,889	0,967	1,560	1,029	1,763	2,121	3,026	4,836	8,003	14,601	27,331
4	0,718	0,874	0,968	1,809	1,077	1,903	2,340	3,151	4,883	8,299	14,587	27,441
5	0,671	0,827	0,998	1,046	1,061	2,293	2,122	3,089	4,805	8,284	14,523	27,440
6	0,733	0,843	0,983	1,029	1,029	2,886	2,168	3,135	4,820	8,268	14,461	27,378
7	0,811	0,921	0,983	1,014	1,045	2,012	2,169	3,120	4,821	8,049	14,555	27,300
8	0,811	0,920	0,983	1,014	1,077	2,278	2,153	3,105	4,805	8,096	14,961	27,394
9	0,968	0,936	0,967	1,029	1,045	5,054	2,137	3,073	4,804	8,128	15,022	27,425
10	0,811	0,873	0,983	1,014	1,045	2,060	2,153	3,120	4,790	8,268	14,758	28,064
11	0,873	0,874	0,967	1,014	1,061	1,544	2,168	3,105	4,836	8,471	14,524	27,519
12	0,905	0,843	0,983	1,014	1,045	1,529	2,152	3,229	4,852	8,222	14,617	27,893
13	1,029	0,967	0,967	1,030	1,061	1,513	2,153	3,120	4,867	8,221	14,570	27,300
14	0,812	0,858	0,951	1,014	1,045	1,513	2,137	3,120	4,883	9,375	14,617	27,316
15	0,904	0,968	0,967	1,029	1,045	1,498	2,106	3,120	4,836	8,034	14,695	27,503
16	0,796	0,873	0,967	0,999	1,061	1,528	2,137	3,089	4,851	8,018	14,758	27,347
17	0,811	0,905	0,967	1,014	1,029	1,513	2,138	3,135	4,868	7,940	14,914	27,284
18	0,780	0,905	0,967	1,014	1,233	1,560	2,137	3,167	4,851	8,019	15,023	27,300
19	0,765	0,936	0,952	1,014	1,544	1,575	2,153	3,136	4,883	8,159	14,445	27,331
20	0,827	0,842	0,968	1,092	1,108	1,514	2,121	3,120	4,774	8,159	14,618	27,300
Průměr [s]	0,8105	0,8900	0,9719	1,10445	1,0873	1,9413	2,1527	3,1161	4,872	8,2111	14,679	27,432
Spojení celkem	1234	1124	1029	905	920	515	465	321	205	122	68	36
Propustnost [kB/s]	123,38	561,76	1296,4	1358,14	1839,41	2060,42	3716,17	5134,62	6568,14	7794,32	8719,55	9331,94

Tabulka 52: C1xS1 - Test LAN 100xConnect-SEND-RECV-FIN

C16xS32 - Test LAN 16x1000xConnect-SEND-RECV-FIN												
	Vláken serveru:			32			Klientů:	16		Počet spojení:		16x1000
	Velikost zprávy / čas testu [s]											
Test č.	**100 B**	**500 B**	**1260 B**	**1,5 kB**	**2kB**	**4 kB**	**8 kB**	**16 kB**	**32 kB**	**64 kB**	**128 kB**	**256 kB**
1	5,054	5,460	5,101	5,382	5,475	7,192	12,152	23,088	45,380	89,825	179,010	357,163
2	5,367	5,461	5,148	5,428	5,476	7,082	12,137	23,478	45,693	89,825	179,229	357,115
3	5,647	5,360	5,054	5,366	5,366	7,082	12,215	23,072	45,287	89,871	178,995	357,147
4	5,491	5,585	5,194	6,037	5,491	7,082	12,121	23,12	45,302	89,872	178,917	357,147
5	5,460	5,928	5,116	5,554	5,444	7,083	12,152	23,182	45,287	89,856	178,979	357,178
6	5,429	5,460	5,070	5,444	5,834	7,820	12,153	23,353	45,396	89,825	178,932	357,179
7	5,460	5,492	5,242	5,662	6,942	7,980	12,137	23,135	45,319	89,840	178,932	357,178
8	5,523	5,445	5,180	5,444	6,256	7,066	12,184	23,073	45,334	89,825	178,994	357,178
9	5,460	5,647	5,102	5,413	5,445	7,129	12,246	23,025	45,614	89,825	178,932	357,178
10	5,366	5,344	5,180	5,569	5,382	7,062	12,153	23,057	45,521	89,872	178,947	357,163
11	5,522	5,460	5,085	5,522	5,398	7,082	12,168	23,025	45,365	89,871	178,932	357,163
12	5,382	5,543	5,038	5,522	5,491	7,098	12,152	23,041	45,333	89,903	178,979	357,147
13	5,522	5,530	5,194	5,538	5,507	7,098	12,137	23,042	45,349	89,903	178,916	357,132
14	5,413	5,610	5,054	5,429	5,491	7,082	12,152	23,026	45,303	89,856	178,933	357,131
15	5,428	5,362	5,085	5,631	5,507	7,082	12,168	23,041	45,318	89,857	178,932	357,132
16	5,397	5,362	5,133	5,428	5,382	7,083	12,137	23,057	45,287	89,888	178,963	357,163
17	5,570	5,400	5,085	5,491	5,523	7,083	12,137	23,057	45,287	89,872	178,948	357,147
18	5,492	5,450	5,163	5,491	5,367	7,098	12,153	23,041	45,318	89,872	178,964	357,147
19	5,538	5,532	5,148	5,413	5,460	7,114	12,152	23,041	45,287	89,903	178,917	357,318
20	5,350	5,530	5,085	5,523	5,460	7,083	12,152	23,042	45,287	89,871	178,947	357,475
Průměr [s]	5,443	5,498	5,124	5,514	5,585	7,174	12,158	23,099	45,363	89,860	178,960	357,179
Spojení celkem	2939	2910	3123	2902	2865	2230	1316	693	353	178	89	45
Propustnost [kB/s]	287,04	1420,1	3842,3	4352,3	5729,8	8921	10528,1	11082,3	11286,6	11395,3	11443,6	11467,6

Tabulka 53: C16xS32 - Test LAN 16x1000xConnect-SEND-RECV-FIN

C16xS32 - Test Lan 16x20xConnect-SEND-RECV-FIN										
	Vláken serveru:			32	Klientů:		16	Počet spojení:		16x20
	Velikost zprávy / čas testu [s]									
Test č.	**0,5 MB**	**1 MB**	**2 MB**	**3 MB**	**4 MB**	**6 MB**	**8 MB**	**10 MB**	**12 MB**	**14 MB**
1	14,289	29,141	59,686	89,216	118,358	175,453	232,206	289,474	346,414	403,447
2	14,321	29,172	59,826	89,544	x	x	x	x	x	x
3	14,305	29,173	60,154	89,544	x	x	x	x	x	x
4	14,290	29,141	59,997	89,356	x	x	x	x	x	x
5	14,305	29,016	60,294	89,511	x	x	x	x	x	x
6	14,289	29,157	60,216	89,606	x	x	x	x	x	x
7	14,305	29,172	60,170	89,511	x	x	x	x	x	x
8	14,305	29,219	60,154	89,606	x	x	x	x	x	x
9	14,305	29,250	60,048	x	x	x	x	x	x	x
10	14,305	29,188	60,513	x	x	x	x	x	x	x
11	14,320	29,157	60,341	x	x	x	x	x	x	x
12	14,305	29,156	59,826	x	x	x	x	x	x	x
13	14,290	29,094	60,029	x	x	x	x	x	x	x
14	14,305	29,156	60,045	x	x	x	x	x	x	x
15	14,305	29,203	60,122	x	x	x	x	x	x	x
16	14,305	29,172	60,029	x	x	x	x	x	x	x
17	14,305	29,109	59,826	x	x	x	x	x	x	x
18	14,305	29,172	60,045	x	x	x	x	x	x	x
19	14,305	29,172	60,232	x	x	x	x	x	x	x
20	14,289	29,266	60,076	x	x	x	x	x	x	x
Průměr [s]	14,303	29,164	60,081	89,487	118,358	175,453	232,206	289,474	346,414	403,447
Spojení celkem	22	11	5	4	3	2	1	1	1	1
Propustnost [MB/s]	11,187	10,972	10,652	10,728	10,815	10,943	11,024	11,054	11,085	11,104

Tabulka 54: C16xS32 - Test Lan 16x20xConnect-SEND-RECV-FIN

C32xS8 - Test LAN 32x100xConnect-SEND-RECV-FIN												
	Vláken serveru:			8			Klientů:	32		Počet spojení:		32x100
	Velikost zprávy / čas testu [s]											
Test č.	**100 B**	**500 B**	**1260 B**	**1,5 kB**	**2kB**	**4 kB**	**8 kB**	**16 kB**	**32 kB**	**64 kB**	**128 kB**	**256 kB**
1	0,920	0,905	0,858	0,873	0,889	1,373	2,433	4,644	9,079	17,987	35,786	71,838
2	0,950	0,936	0,858	0,873	0,874	1,388	2,434	4,649	9,064	17,971	35,802	71,853
3	0,936	0,952	0,858	0,874	0,889	1,388	2,434	4,649	9,063	17,987	35,786	71,838
4	0,906	0,920	0,842	0,842	0,889	1,404	2,465	4,649	9,064	17,971	35,787	71,760
5	0,902	0,921	0,811	0,858	0,936	1,389	2,496	4,602	9,063	17,987	35,802	71,807
6	0,920	0,900	0,827	0,874	0,921	1,388	2,433	4,633	9,064	17,972	35,802	71,807
Průměr [s]	0,922	0,922	0,842	0,865	0,899	1,388	2,449	4,637	9,066	17,979	35,794	71,817
Spojení celkem	3469	3469	3799	3697	3557	2305	1307	690	353	178	89	45
Propustnost [kB/s]	338,9	1694,1	4674,5	5544,9	7113,8	9219,7	10452,5	11040,03	11294,74	11390,9	11443,2	11406,7

Tabulka 55: C32xS8 - Test LAN 32x100xConnect-SEND-RECV-FIN

C32xS12 - Test LAN 32x100xConnect-SEND-RECV-FIN												
	Vláken serveru:		12		Klientů:		32		Počet spojení:		32x100	
	Velikost zprávy / čas testu [s]											
Test č.	**100 B**	**500 B**	**1260 B**	**1,5 kB**	**2kB**	**4 kB**	**8 kB**	**16 kB**	**32 kB**	**64 kB**	**128 kB**	**256 kB**
1	0,921	0,920	0,874	0,921	0,905	1,404	2,434	4,633	9,063	17,972	35,787	71,51
2	0,889	0,920	0,873	0,905	0,89	1,419	2,433	4,649	9,064	17,971	35,802	71,500
3	0,920	0,905	0,858	0,905	0,905	1,388	2,418	4,617	9,079	17,987	35,802	71,512
4	0,874	0,905	0,904	0,889	0,889	1,419	2,418	4,633	9,079	17,971	35,833	71,526
5	0,909	0,904	0,858	0,843	0,921	1,388	2,434	4,634	9,282	17,956	35,786	71,510
6	0,905	0,890	0,842	0,905	0,889	1,450	2,430	4,633	9,064	18,002	35,802	71,510
7	0,920	0,904	0,904	0,889	0,889	1,450	2,418	4,634	9,079	17,987	35,833	71,510
8	0,910	0,901	0,858	0,884	0,889	1,419	2,433	4,617	9,079	18,002	35,802	71,512
Průměr [s]	0,906	0,906	0,871	0,892	0,897	1,417	2,427	4,631	9,098	17,981	35,805	71,512
Spojení celkem	3532	3532	3672	3585	3567	2258	1318	691	352	178	89	45
Propustnost [kB/s]	344,9	1724	4519	5377	7134	9032	10545	11055	11254	11390	11439	11455

Tabulka 56: C32xS12 - Test LAN 32x100xConnect-SEND-RECV-FIN

C32xS16 - Test LAN 32x100xConnect-SEND-RECV-FIN												
	Vláken serveru:			16	Klientů:			32	Počet spojení:			32x100
	Velikost zprávy / čas testu [s]											
Test č.	**100 B**	**500 B**	**1260 B**	**1,5 kB**	**2kB**	**4 kB**	**8 kB**	**16 kB**	**32 kB**	**64 kB**	**128 kB**	**256 kB**
1	0,920	0,920	0,843	0,921	0,905	1,388	2,450	4,649	9,094	17,986	35,802	71,744
2	0,905	0,889	0,889	0,873	0,889	1,373	2,434	4,633	9,079	17,987	35,802	71,448
3	0,889	0,920	0,827	0,889	0,905	1,404	2,496	4,634	9,064	17,987	35,802	71,464
4	0,874	0,889	0,842	0,858	0,889	1,388	2,434	4,649	9,094	17,986	35,802	71,744
5	0,905	0,920	0,827	0,842	0,889	1,388	2,450	4,633	9,079	17,987	35,802	71,448
6	0,904	0,874	0,904	0,889	0,905	1,373	2,434	4,634	9,064	17,987	35,802	71,464
Průměr [s]	0,899	0,902	0,855	0,879	0,897	1,386	2,450	4,639	9,079	17,986	35,802	71,552
Spojení celkem	3558	3548	3741	3642	3567	2309	1306	690	352	178	89	45
Propustnost [kB/s]	347,42	1732	4604	5463	7135	9237	10450	11038	11279	11386	11441	11449

Tabulka 57: C32xS16 - Test LAN 32x100xConnect-SEND-RECV-FIN

C32xS32 - Test LAN 32x100xConnect-SEND-RECV-FIN												
	Vláken serveru:			32		Klientů:		32		Počet spojení:		32x100
	Velikost zprávy / čas testu [s]											
Test č.	**100 B**	**500 B**	**1260 B**	**1,5 kB**	**2kB**	**4 kB**	**8 kB**	**16 kB**	**32 kB**	**64 kB**	**128 kB**	**256 kB**
1	1,076	1,123	0,890	1,029	0,999	1,389	2,418	4,649	9,095	17,987	35,802	71,511
2	1,070	1,153	0,968	0,983	1,061	1,388	2,149	4,633	9,111	18,018	35,802	71,479
3	1,092	1,108	0,952	0,998	1,014	1,388	2,434	4,633	9,095	18,002	35,802	71,510
4	1,108	1,076	0,983	0,952	1,045	1,388	2,433	4,634	9,095	18,003	35,849	71,448
5	1,107	1,061	0,889	1,014	1,076	1,373	2,449	4,649	9,079	18,018	35,833	71,433
6	1,108	1,060	0,921	0,952	1,014	1,388	2,418	4,680	9,095	18,007	35,817	71,433
Průměr [s]	1,096	1,097	0,934	0,990	1,035	1,390	2,384	4,646	9,095	18,006	35,818	71,470
Spojení celkem	2926	2917	3427	3239	3092	2309	1343	689	352	178	89	45
Propustnost [kB/s]	285,8	1424	4216	4858	6184	9237	10740	11019	11258	11374	11435	11462

Tabulka 58: C32xS32 - Test LAN 32x100xConnect-SEND-RECV-FIN

C32xS32 - Test LAN 32x20xConnect-SEND-RECV-FIN										
	Vláken serveru:		32		Klientů:		32		Počet spojení:	32x20
	Velikost zprávy / čas testu [s]									
Test č.	**0,5 MB**	**1 MB**	**2 MB**	**3 MB**	**4 MB**	**6 MB**	**8 MB**	**10 MB**	**12 MB**	**14 MB**
1	28,782	58,453	117,187	174,253	231,271	344,729	458,641	573,005	687,618	801,358
Průměr [s]	28,78	58,45	117,19	174,25	231,27	344,73	458,64	573,00	687,62	801,36
Spojení	22	11	5	4	3	2	1	1	1	1
Propustnost [MB/s]	11,118	10,949	10,92	11,019	11,069	11,139	11,163	11,169	11,169	11,181

Tabulka 59: C32xS32 - Test LAN 32x20xConnect-SEND-RECV-FIN

F Obsah přiloženého DVD

- /src - Obsahuje zdrojové kódy.
  - /cpp - Použité kódy v jazyce C++.
    - /SDDS_v2.0
      - /common/networking - třídy pro práci se sockety
      - /dbms/clientserver - třídy serveru a klienta
      - /dbms/ddbms - třídy s mapováním a pohledem na DDS
  - /Embedded
    - /EmbeddedBtree - aplikace s testy DS B-stromu.
    - /EmbeddedRtree - aplikace s testy DS R-stromu.
  - /MutexMeteredSection
    - /2.3 Mutex - test synchronizace za pomoci Mutexu.
    - /2.3 MeteredSection - test synchronizace za pomoci Metered Section.
  - /SDDS_v1.0 - prvotní verze aplikace.
  - /SDDS_v2.0 - finální verze aplikace.
- /text - obsahuje text diplomové práce v Latexu.
- /log - obsahuje záznamy a nastavení serverů z testů.
  - /virt_23
    - /B5 - test s B-stomem a 5-ti uzly.
    - ...
    - /R20 - test s R-stomem a 20-ti uzly.
  - ...
  - /virt_102
- /TCP Report - záznamy meření TCP výkonu
  - /cz - česky psaná verze
  - /en - anglicky psaná verze